COLORCUE

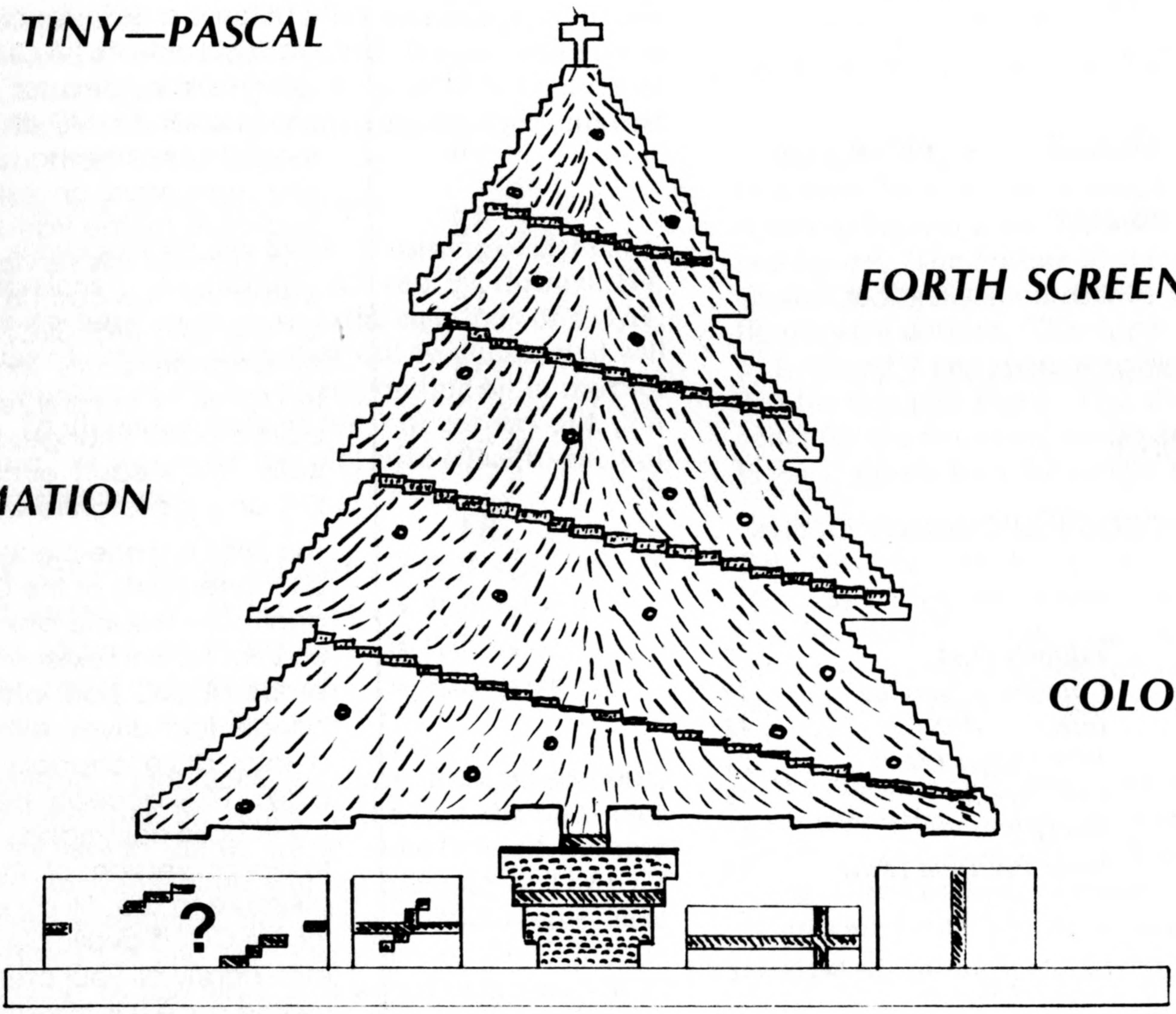
VOLUME VI
NUMBER 5

A BI-MONTHLY PUBLICATION BY AND FOR INTECOLOR AND COMPUCOLOR USERS

TINY—PASCAL

FORTH SCREEN EDITOR

ANIMATION



COLORWORD

ZIP

HEX TO ASCII CONVERSION

Colorcue

VOLUME VI, NUMBER 5 SEPTEMBER/OCTOBER 1984

CONTENTS

EDITOR: JOSEPH NORRIS COMPUSERVE: 71106, 1302

3651 PRICE REDUCTION

Intecolor Corporation has announced a price reduction on the 3651 computer. With 32K memory, deluxe keyboard, one 90K internal disk drive, manual and sampler disk, this excellant computer is available for $1450.00. The 3651 carries FCS v9.80, which is nearly identical to v8.79 of the CCII. [COLORCUE uses a 3651 for all its editing and data base work.] A CDBKUP.COM program is provided for transferring CD programs to the MD disk format used by the 3651. In order to use this program you must have an external CD drive available for the transfer. [COLORCUE can provide this service for you if you wish at very low cost.] In our experience, all files with BAS, SRC, and TXT extensions transfer easily. Most PRG files written for v8.79 transfer readily. A few require modest changes to esoteric ROM calls. This is easily accomplished with IDA and is straightforward.

The 3651 is a one-piece computer, with all the features of the CCII, plus faster and more reliable disk reads, improved internal and external commands, a proper RS232 port, and the ability to handle four drives, either 5" or 8", for a total drive capacity of 2.4 megabytes. FCS accepts both upper and lower case commands. It is a truly professional version of the CCII and a pleasure to use. All important software for the CCII is available for the 3651. In fact, nearly all your present software is usable once it is transferred to the MD disk. The new price includes your selection of 10 software packages from the Intecolor catalog. Intecolor Corporation. 225 Technology Park, Norcross, GA 30092. (404)-449-5961.

COLORCUE is published bi-monthly. Subscription rates are US$12/year in the U.S., Canada, and Mexico (via First Class mail), and US$30 elsewhere (via Air Mail). All editorial and subscription correspondence should be addressed to COLORCUE, 19 West Second Street, Moorestown, NJ 08057, USA. (609-234-8117) All articles in COLORCUE are checked for accuracy to the best of our ability but cannot be guaranteed error free.

from the Editor's Desk

Anything involving computers is in a constant state of flux these days, and we at Colorcue aren't exempt from the trend. While we are 200 subscribers strong, we are rather weak in terms of the kind of activity that keeps an expensive magazine such as Colorcue alive and energetic. In short, we have had to accept the unavoidable truth that we cannot continue to publish in our present format.

Volume VI will be the last for Colorcue (one issue to come) and we will be combined with CHIP, the newsletter of the Rochester User Group, under the guidance of Rick Taubold. The details are explained elsewhere in this issue, so please take note of them, and lend your support to this splendid group as they continue their long history of Compucolor activity. I will continue as an author for CHIP, and have offered my support as my time and means permit.

Since my work keeps me at one keyboard or another most of the time (on three different operating systems!) I share the turmoil felt by most of the industry on a day-to-day basis. With the avalanche of technological sophistication and the increased demands of business on software efficiency, those of us who indulge in computer interests as avocation are finding a major change in the way we must approach our hobby, principally in the ways we differentiate between hobby and our business activity.

Small computers are great fun, relaxing to operate, and elegant learning tools. Most of the commercial "small" machines are rapidly disappearing as bankruptcy claims its victims left and right. Those of us accustomed to being in the vanguard of computer experimentation and development, pounding keyboards late into the night, assembling equipment from surplus houses, and pushing forth the frontiers of technology, are now replaced by the huge engineering teams and multi-billion dollar resources of industrial giants. The new technology belongs to the rich and powerful.

While we have lost our mainstream importance as hobbyists, I greet this change as a positive one overall. The hobby can now flourish as a hobby without the oppression of changing business needs and economic necessity. While we will continue to benefit from the reduced prices of obsoleted equipment and software, we will also be banding together as a computer-interest community, consolidating our resources, much as we did in the beginning; sharing more, learning more, and contributing more. Expect to see new magazines devoted to the hobby of computer electronics, such as the proposed hardware periodical, Computersmith, to be launched in the Spring of 1985 by Ed Dell in New Hampshire. (Ed publishes The Audio Amateur and Speaker Builder, two very fine magazines for audio enthusiasts.) Expect to see more magazines for the amateur devoted to experimental programming philosophies, and an ever increasing proliferation of public-domain software. Relieved from the pressures of profit making, many talented people will find more time to devote to the avocational aspects of their interests.

It is important to realize that no computer is really obsolete in the face of this kind of activity. Much as the humble Sinclair computer challenged many professionals to overcome its limitations, so can we pursue the challenge of implementing many new and fascinating procedures on the 8080 and Z80 machines (such as the "window", pull-down menu, and multi-tasking.) Experimenting, at least on a small scale, is easier on a simple computer with a straightforward operating system (and FCS is ideal for this!). Programming is quick and easily changed.

Many of you have not yet tapped the abundance of excellant programming tools available from the CHIP library, such as Pascal, Forth, Tiny-C, and the FASBAS and ZIP compilers. Colorcue subscribers have written a word processor with spelling checker, an analog circuit analysis program complete with Bode plots, and several very clever disassemblers. We have in our software bank the best programs ever written for program development for any computer, including editors, assemblers, disassemblers, and compilers. It is no longer appropriate to look only for programs written specifically for the Compucolor. We are equipped to do almost anything any other computer can do. I would like to see programs to emulate the batch files of MSDOS, and the power of dBASE II on the Compucolor. They are within our grasp.

If you are planning to purchase new equipment, and have given your best effort to the CCII, you are in a very fortunate position to make wise choices and advance your skills. Few computer owners have been in a position to get as much for their investment as we have, even without factory support.

Intecolor has been in communication with Colorcue and CHIP in the last months with offers of a renewed effort to be of service to CCII owners. While there isn't a great deal they can do at this late date, we appreciate their interest and spirit of good will. We appreciate their acknowledgement of our independent achievements as a community.

You will notice the announcement of a new compiler by Peter Hiner in this issue. There is irony and humor in the fact of its appearance in this particular issue of Colorcue. I remember very well that FASBAS was first reviewed in the final issue of FORUM several years ago.

So as we prepare to bid "farewell" to our present format, we are also proceeding with continued energy to improve the resources for the CCII, and looking forward to an expanded CHIP. We extend our best wishes, and our appreciation, to this organization, for coming to the rescue still another time.

This issue has been greatly delayed for lack of sufficient material to fill its pages. The final issue will be subject to the same restraints, so I urge you to submit your materials as promptly as you can. I would like to see the Nov/Dec issue long before the Spring of 1985, and I will need your help to achieve that goal. My wife, Susan, and I extend our best wishes to you all for a happy and fulfilling New Year. □

Joseph

PETER HINER *11 Penny Croft, Herpenden, Herts, AL5 2PD, ENGLAND*

A NEW COMPILER CALLED

I expect that most of you already know of my compiler FASBAS, which speeds up Basic programs. Those of you who have used FASBAS will know that it is easy to use, that it accepts Basic programs with virtually no restrictions, and that speed increases of up to five times can be achieved. Now I have produced a new compiler called ZIP, which is not as easy to use and which imposes some restrictions, but which can make programs run much faster than when compiled by FASBAS.

ZIP achieves much greater speed by treating all variables and constants as integers (instead of using floating-point arithmetic like Basic and FASBAS). This means that ZIP is not suitable for all applications, and that Basic programs may need to be modified to suit the requirements of integer arithmetic. The manual supplied with ZIP explains how to modify your programs, and the disk includes an Integer Basic Interpreter with debugging facilities. So you should be well equipped to overcome any problems, and you will find that ZIP opens the way to using Basic for fast graphic displays and arcade games.

A review in the September, 1982 issue of BYTE included some benchmark programs for comparing the relative speed increases achieved by integer and floating-point compilers for the Apple computer. The table in Fig 1. compares ZIP and FASBAS results with the ranges of results for Apple compilers. The figures in the table are the approximate number of times faster than normal Basic for each benchmark program. The ZIP results for benchmark programs 7 and 8 are bracketed because integer arithmetic does not give correct answers. (One of the Apple integer compilers handled this situation by reverting to floating-point arithmetic.)

You can see that, while FASBAS comes somewhere in the middle of the field of floating-point compilers, ZIP beats them all.

You can use ZIP to compile existing Basic programs. Many will work with little or no modification, some will require care and patience, and others simply will not be able to give the required results using integer arithmetic. However, the best way to use ZIP is to consider it as a tool, which allows you to use familiar Basic functions for the framework of your program, while exploring techniques similar to those used in Assembly Language programs to optimize the speed of critical routines.

The ZIP manual describes an example where the PRINT function is replaced by POKE to display a row of stars on the screen. The times for 1000 executions of this routine are shown in Fig 2, in seconds.

The POKE method gives ZIP an 18-fold speed improvement over the normal Basic PRINT method. To see how this compared with Assembly Language (machine code), I wrote an Assembly Language routine which took 3.3 seconds for 1000 executions. With further optimization of the ZIP routine I could cut the time to seven seconds, achieving half the speed of machine code (and more than 26 times the speed of Basic) for this routine. This shows that fast graphic displays and arcade games in Basic are really made possible by ZIP.

How, then, should you treat ZIP? Should you consider it as another Basic compiler, which can produce much faster results once you have overcome the obstacles of integer arithmetic? Or should you treat it as a new way of writing programs, almost like using a new language which you nearly know already? Whichever way you choose, ZIP offers you the excitement of holding a tiger by its tail.

The minimum requirements for running ZIP are 16K of user memory, one disk drive, and v6.78, v8.79, or v9.80 Basic. The programs compiled by ZIP are generated as PRG files for convenience of operation, and will automatically run on v6.78, v8.79, or v9.80 machines.[*]

I propose to distribute ZIP in the same way as I distributed FASBAS, encouraging user groups to make the program available to their members on payment of a fee of $15.00 per copy. Individuals are welcome to order direct from me, in which case the price is $25.00 for the disk (in CCII format), manual and airmail postage. Personal

Fig 1.

BENCHMARKS	1	2	3	4	5	6	7	8
FASBAS	33	24	29	16	28	8	(5)	(17)
Apple Integer	11-16	9-15	7-14	5-15	8-17	8	5	4
ZIP	4	4	3	3	3	2	2	4
Apple F. P.	3-6	2-4	2-4	2-4	2-5	2-3	2-5	3-5

zzzzzzipppppp

checks are welcome as the method of payment.

The deal is further complicated by an offer of the latest version (v12.24) of FASBAS, which now includes operation on machines with v9.80 Basic, output files in PRG form (instead of LDA) and compiled programs which will automatically run on v6.78, v8.79, or v9.80 machines.[*] The new version also fixes a few obscure bugs and makes some improvements to the size and speed of compiled programs.

Using	BASIC	FASBAS	ZIP
PRINT	187	143	82
POKE	346	132	10

Fig 2.

Previous purchasers of FASBAS will get a free update included on the ZIP disk. New purchasers may order the pair of compilers (ZIP and FASBAS) on one disk for a combined price of $40.00 from me (or a fee of $30.00 if obtained through a user group). As always, I will send a free replacement for any disk which is unreadable or damaged in transit. □

[* This means, for example, that a program written in v6.78 will run on any other version of FCS without a jump table or conversion of any kind. ED.]

HEX TO ASCII CONVERSION (intecolor 8000)

BOB MENDELSON 27 Somerset Place, Murray Hill, NJ 07974

Whenever the ISC 8001 computer is used for BASIC programming there is a need for decimal equivalents to the hex memory addressses used for PEEK and POKE statements. While there are tables for these conversions, unless the table is exceedingly long, interpolation is required. It is true that Texas Instruments makes a hand calculator for direct hex-decimal-hex conversions, but even here it is necessary to make a manual subtraction of 65536 if the original hex number is greater than 8000. This is because the ISC ROM requires that numbers greater than 8000H be used in Basic as 65536—hex#. This program will take any hex number as input, convert and print it as a decimal number, including numbers greater than 8000H. The program is only 180 bytes long and is well worth adding to your general utility disk.

The initial steps from START to HEX set up the title and column names. At HEX, the routine HEXIN calls from ROM a program that takes the ASCII input for a hex number and converts it to a true hex number which is stored in memory at ORIG.

The standard hex to decimal routine STDEC is then called. It first loads 5 bytes of memory defined as UNIT, TEN, HUN, THOU and TTHOU designating each of the 5 digits in the decimal number. The value 30H that is loaded is the ASCII value for zero. As we shall see, this is very important.

Now the actual conversion starts. By use of the 2's complement method, starting with the most significant digit, successive additions are made of the hex equivalent of 10000 decimal. Each addition is counted by adding one to the TTHOU memory byte. This raises ASCII 30H to successive ASCII numbers. This continues until a borrow occurs and the carry is 'true.' The same procedure is followed for each of the next three digits. The unit digit is converted by adding it to 30H and saving it.

The program is now ready to print the result under the decimal column, beginning at routine PRINT. Register C is set to 5 as a counter. Register B is set to 30H as a detector of unneeded zeros to the left of small numbers, and register D is set to zero as a detector of the first digit greater than zero that is printed. This will be used to retain the print of zeros in the middle of a number.

The most significant digit is recovered and compared with 30H. If it is not a zero, it is printed and the D register is set to 1. The next digit is recovered and the same sequence is followed. A zero recovered while D = 0 will be replaced by a space. When D = 1, the zero will be printed. The standard decimal conversion process is now over.

The conversion to the ISC decimal equivalents requires that the input number be checked to see if it is greater or less than 8000H. This is done at DSPASC. If less than 8000H, the print will be the same as described above. If it is greater than 8000H, then 65536 will have to be subtracted to get the ISC minus number. A modified 2's complement method is used to achieve this. Since the hex value for 65536 is 10000H, it is too large to use 16 bit arithmetic. Therefore a 24 bit routine is used. Further, the need for a minus sign can be met by merely printing it, since we won't be in this portion of the software if it is not required. The routine starts at NEG, where '00' is added in front of the input hex number (1 byte) and the constant 010000H is provided by loading three bytes. It is time, now, to perform the subtraction-with-carry on the least significant byte. The result replaces the input hex number at the ORIG memory address.

As a result of all of this arithmetic, the ORIG address now holds the hex equivalent of the ISC negative number. The STDEC routine is recalled to convert this new number to 5 ASCII digits which are then printed in the ISC column. The number 8000H cannot be defined since it is neither positive nor negative, so a routine at ISC is used to detect this number and print a 'Not Defined' error message.

The program now recycles and asks for a new input. Any non-hex character at the input will cause a jump out of the program. This may be used for an orderly escape, including a vector to a utility menu. □

```
;LISTING I. HEXASC.SRC - Hex to ASCII Conversion

;          For ISC 8001 Computer
;          by R. M. Mendelson, 7/8/84

;          Program origin = FE00H

;Equates ......................................

        CI      EQU     0103H   ;Console in
        CO      EQU     0109H   ;Console out
        EXPR    EQU     0133H   ;Hex evaluation
        KEYBF1  EQU     9FFFH   ;Keyboard flag
        LO      EQU     010FH   ;List out
        OSTR    EQU     012AH   ;Output string
        UTIL    EQU     5006H   ;Utility ROM, may
                                ; be any address
        BA7ON   EQU     14
        BA7OFF  EQU     15
        CR      EQU     13
        CYAN    EQU     22
        EL      EQU     11
        EOS     EQU     239
        EP      EQU     12
        GREEN   EQU     18
        LF      EQU     10
        RED     EQU     17
        UP      EQU     28
        YEL     EQU     19

        ORG     FE00H

START:  XRA     A
        STA     KEYBF1  ;Clear keyboard
        CALL    MESS
        DB      YEL,BA7ON,EP
        DB      'HEX TO ASCII DECIMAL CONVERSIONS-'
        DB      BA7OFF

        ;Column labels

        DB      CR,LF,LF,GREEN,'  HEX      STD     ISC'
        DB      CR,LF,RED,' ----       ',YEL,' -----  '
        DB      CYAN,'  -----',CR,LF,EOS

        ;Main Routine. Input will appear in the HL
        ;             register as hex #

        JMP     REPEAT  ;Get input arrow
HEX:    CALL    HEXIN   ;Input kybd ASCII input to hex#
        SHLD    ORIG    ;Save it for later use
        CALL    STDEC   ;Convert to std ASCII decimal #
        CALL    MESS    ;Align cursor for dec printout
        DB      CR,LF,'         ',YEL,UP,EOS
        CALL    PRINT   ;Print it out

;This routine will check if number is greater than 8000H
; and if so will subtract 65536 and print it with a minus
; sign as required by the ISC 8001. If the number is less
; than 8000H, it will print it as standard.
        LHLD    ORIG    ;Recover original hex# from mem.
DSPASC: XRA     A       ;Set A=0
        DCR     A       ;A=FFFFH
        ANA     H       ;If H=) 8xxx, sign = 1 or minus
        JM      ISC     ;If minus print '-' sign
        CALL    PRINT   ;Print it. Dup of col 1( 8000H
        JMP     REPEAT  ;Ready for new input

;Check for input of 8000H which is not defined as (-)#
ISC:    MOV     A,H     ;Get most significant byte
        SUI     80H     ;If H=80 ANS=0, zero flag=1
        ADD     L       ;Any digit but 00 resets zero fl
        JZ      OVFLOW  ;Number is 8000H
;Print minus sign and perform subtraction
        MVI     A,26    ;Cursor left for '-' sign
        CALL    LO      ;Move it
        MVI     A,'-'   ;Minus sign
        CALL    LO      ;Print it
        CALL    NEG     ;Subtract 65536
        CALL    STDEC   ;Print difference as dec #
        CALL    PRINT   ;Print it
        JMP     REPEAT  ;Ready for new input


;Conversion to std decimal number. Fill storage
; memory with ASCII 0 (30H)

STDEC:  LHLD    ORIG    ;# to be converted must be
        XCHG            ; in DE to start
        LXI     H,UNIT  ;Point to 5 digit storage
        MVI     C,5     ;Counter
SETTOO: MVI     M,30H   ;Fill with ASCII 0
        INX     H
        DCR     C       ;Count down
        JNZ     SETTOO  ;Re-do

;Convert hex to ASCII (0-65536)

        DCX     H       ;Back up to ten-thou digit.
        LXI     B,0D8F0H ;-10000 (2's comp of 10K)
        CALL    DIGIT   ;Add to the content of DE
                        ; reg until borrow occurs.


        LXI     B,0FC18H ;-1000 (2's comp)
        CALL    DIGIT   ;Do again til borrow occurs

        LXI     B,0FF9CH ;-100 (2's comp)
        CALL    DIGIT

        LXI     B,0FFF6H ;-10 (2's comp)
        CALL    DIGIT   ;Final addition
```

```
          MOV     A,E     ;Get the number of units
          ADI     30H     ;Add ASCII 0 to it
          MOV     M,A     ;Save it.
          RET

;Print out the ASCII standard decimal #

PRINT:    LXI     H,TTHOU ;Point HL to MSB ten-thous
          MVI     C,5     ;Set counter: print 5 digits
          MVI     B,30H   ;ASCII 0 (LXI B,3005H)
          MVI     D,0     ;Marker for digit or zero
PR0:      MOV     A,M     ;Get ASCII digit
          CMP     B       ;Is it ASCII zero?
          JNZ     PR1     ;No! Set marker
          XRA     A
          CMP     D       ;Yes! Check marker for 0
                          ; No digits yet!
          JNZ     PR2     ;Marker is '1'. Print 0
          MVI     A,20H   ;Marker is '0', so
          JMP     PR3     ; print space
PR1:      MVI     D,1     ;Digit for print, set marker=1
PR2:      MOV     A,M     ;Recover non-zero digit
PR3:      CALL    CO      ;Print it
          DCX     H       ;Dec pntr to next sig. digit
          DCR     C       ;Dec counter
          JNZ     PR0     ;Re-do till counter = 0

;Set cursor for last # printout
          CALL    MESS
          DB      '      ',CYAN,EOS
          RET

;Subtract 65536 to get ISC negative number. Must
; get input number and 2's comp of FFFF in 3 bytes
; each into memory, in order LSB, NSM, MSB. The
; 2's complement of 65536 is -10000d.

NEG:      LXI     D,MINUS
          XRA     A
          STA     MINUS   ;MINUS has 10000
          STA     MINUS+1
          STA     ORIG+2  ;Blank MSB for 3rd byte of
                          ; input number.
          INR     A
          STA     MINUS+2 ;The '01' of '010000'

SUB3:     MVI     C,3     ;Counter
          LXI     H,ORIG  ;Point to lsb of original #
          XRA     A
SUBAGN:   LDAX    D       ;Get lsb of constant
          SBB     M       ;Subtract lsb
          MOV     M,A     ;Replace ORIG digit
          INX     D       ;Next significant bits
          INX     H
          DCR     C       ;Reduce counter
          JNZ     SUBAGN  ;Not done. Re-do, or
          RET             ; return

;Get ready for next hex input
REPEAT:   CALL    MESS    ;Set arrow prompt
          DB      CR,LF,LF,YEL,' > ',EOS
          JMP     HEX

;Routines for 0-65536 decimal conversion

DIGIT:    PUSH    H       ;Save mem pointer on stack
          XCHG            ;Get # for conver. in HL
          DAD     B       ;Add 2's comp test #
          JNC     ADDIT   ;If borrow then carry=True
          XCHG            ;No borrow, get HL & DE back
          POP     H       ;Recover memory address
          INR     M       ;Incr mem count by 1
          JMP     DIGIT   ;Try subtraction again. Loop
                          ; will cont until carry=True
ADDIT:    MOV     A,C     ;Form 2's complement
          CMA
          MOV     E,A
          MOV     A,B
          CMA
          MOV     D,A
          INX     D       ;DE contains 2's comp
          DAD     D       ;Add it to test #
          XCHG
          POP     H       ;Pop mem pointer off stack
          DCX     H       ;Decrement memory pointer
          RET             ;Back to main program

;Subroutines

HEXIN:    MVI     C,1     ;Counts 4 bytes of ASCII input
          CALL    EXPR    ; of a hex# and converts to
          POP     H       ; true hex. Result in HL reg.
          RET

OVFLOW:   CALL    MESS
          DB      RED,'NOT DEFINED',EOS
          JMP     REPEAT

MESS:     POP     H       ;[This is very tricky! Seasoned
          CALL    OSTR    ; programmers, take note! ED]
          PCHL

;Memory address for 0-65536 decimal output

UNIT:     DS      1       ;5 digit memory storage
TEN:      DS      1
HUN:      DS      1
THOU:     DS      1
TTHOU:    DS      1
ORIG:     DS      3       ;Original input hex# + 00 msb
MINUS:    DS      3       ;2's complement of 65536=10000d

          END     START
```

REFERENCES: 8080 Software Design, Book 1. H. W. Sams, 1978, p 257. Computer Design, April 1978, p 168.

........a new FORTH screen editor.......

TOM NAPIER *12 Birch Street, Monsey, NY 10952*

In early 1982 I decided that I would like to try FORTH on my Compucolor II, so I did the logical thing. I bought a listing of FIG-Forth, typed in some 28000 characters, and spent several weekends adapting CP/M based code to the idiosyncrasies of the CCII disk system. Eventually it worked, and it wasn't until several months later that I discovered the DATACHIP library already contained a version of Forth for the CCII. [1]

I found the FIG-Forth editor to be so slow and cumbersome that I often ended up writing programs in assembler rather than in Forth. Recently I took the plunge and wrote my own screen-based editor. It is such a delight to use that I even find myself touching up the layout of Forth screens for fun, something I would never have dreamed of doing before.

Since this editor is written in standard FIG-Forth, it should work with any version of CCII Forth. It should also be Intecolor model independent. Since it uses the editing keys of the expanded keyboard it won't be so easy to use with only the standard keyboard. The allocation of screen numbers in the accompanying listing is quite arbitrary and can be changed to suit the user's disk space, provided the order of the definitions is not changed.

This editor uses one main word and two auxiliary words. The latter two are CLEAR and COPY, which are copied from the FIG-Forth editor. Their usage is 'N CLEAR' to clear disk screen N, and 'N1 N2 COPY' to copy screen N1 to screen N2. The editor is invoked by typing "SEDIT" (for Screen EDITor). At run time it prints the message "NEXT SCREEN NUMBER" and waits for the user to type in the number of the screen to be edited. I've been lazy and omitted the backspace function from the number input, so you'll have to get it right the first time. Type an out-of-range screen number if you want to get a disk error message, and therefore abort a typing mistake.

The specified screen is loaded from disk and displayed on lines 4 through 19. Above these lines is a display of the screen number and the current cursor position. The cursor appears as a blue background travelling under the green ASCII characters. Under the screen, the prompt "PAD", followed by a right arror, appears. This indicates that the line below, initially blank, is a display of the contents of the pad.

One may now type in any characters and have them inserted at the current cursor position. Existing characters on the same line will be moved to the right to make room for the insertion. Any characters moved off the end of the 64 character line will be lost. Note that the insertion process takes a macroscopic time. I occasionally lose one of a double letter pair, and if you are a blindingly fast touch typist you will have to slow down a little.

The DELETE CHAR key deletes the character to the left of the cursor and moves the remaining characters in the line to the left. Lines are treated independently; deleting characters on one line does not move up those on the line below, useful though this might sometimes be.

The cursor keys move the blue cursor around the screen. They auto-repeat and also wrap around at the sides and the top and bottom of the screen. The quickest way from the extreme right of a line to the extreme left is to use the right arrow key. The HOME key returns the cursor to the top left corner.

The RETURN key acts as a "new line" key, moving the cursor to the first left position on the line below.

Complete lines may be manipulated with the ERASE LINE, INSERT CHAR, DELETE LINE, and INSERT LINE keys. INSERT LINE moves the text downward starting with the line the cursor is on, thus enabling one to insert a new line at that point. The last line on the screen is lost in this case. DELETE LINE copies the line the cursor is on to the pad, appearing under the screen display of text, then moves all following lines up one line. ERASE line copies a line to the pad without deleting it. The INSERT CHAR key is used as a 'paste' key; it moves the text from the cursor down to make room for a new line, then copies the text from the pad to the screen. This can be used to transfer text between screens as well as to duplicate lines on the same screen.

ERASE PAGE is used to change from one screen to another. It generates the 'next screen number' prompt and displays the contents of the screen specified. The current screen is flagged as UPDATEd whenever any changes are made to it.

Pressing the ESC key FLUSHes the changes to disk and exits from the editor to FORTH.

One thing to watch—the DELETE CHAR command deletes to the left of the cursor so it cannot erase the last character on the line without one other character being erased first. Also, if one attempts to insert beyond the last character the keyboard input will replace the last character on that line. None of this should matter since the last character of a line should always be a space, otherwise the FORTH compiler will concatenate it with the first character of the next line and be unable to find the resulting word. It took me some minutes of gazing at the message ";:?" before I realized that a FORTH screen contains nothing to indicate where a line ends. (Another tip from hard experience; if a screen contains only a few lines, end them with ";S". If there are more than 255 spaces between the last character and the end of a screen the compiler will hang up.) [Thank you, Tom! I have been scratching my head! ED].

In this listing I've defined four words; CLRS, PLOT, SMOVE, and VHTAB, which my system has built in. This means that this published version displays the screen more slowly than my original version. Happy editing. □

[1] Two versions of FORTH, and three screen editors, to date. [ED]

SCR # 40

```
( SCREEN EDITOR, 5/1/85   COPYRIGHT 1985 T. M. NAPIER )

 6 LOAD  ( CASE FUNCTION )
41 LOAD
42 LOAD
43 LOAD
44 LOAD
45 LOAD
46 LOAD
47 LOAD
;S
```

SCR # 41

```
( SCREEN EDITOR  INSERT DELETE )

0 VARIABLE L#   0 VARIABLE C#   0 VARIABLE K#

: PLOT        64 * + DUP + 28672 + ; ( SCREEN ADDRESS )
: LINE        SCR @ (LINE) DROP ; ( GET LINE ADDRESS )
: CHAR        L# @ LINE C# @ + ; ( CHARACTER ADDRESS )
: -MOVE       LINE C/L CMOVE UPDATE ;  ( MOVE LINE UP ONE )
: WIPE        LINE C/L BLANKS UPDATE ; ( CLEAR LINE )
: CLEAR-PAD   C/L PAD C! PAD 1+ C/L BLANKS ;
: <SLIDE      CHAR DUP 1 - C/L C# @ - CMOVE ;
: SLIDE>      CHAR DUP C/L + C# @ - 1 -
                DO I 1 - C@ I C! -1 +LOOP ;
: DASHES      0 SWAP PLOT DUP C/L 2 * + SWAP
                DO 45 I C! 2 +LOOP ;
```

SCR # 42

```
( SCREEN EDITING COMMANDS )

: CLEAR   SCR ! 16 0 DO I LINE C/L BLANKS LOOP UPDATE ;

: COPY    B/SCR * OFFSET @ + SWAP B/SCR * B/SCR OVER + SWAP
            DO DUP I BLOCK 2 - ! 1+ UPDATE
            LOOP DROP FLUSH ;

: SMOVE   ROT SWAP OVER + SWAP DO   DUP I C@ SWAP C! 2+
                                LOOP DROP ;
: COLOR-CURSOR   C# @ L# @ 4 + PLOT 1+ C! ;
: MARK           34 COLOR-CURSOR ;
: -MARK           2 COLOR-CURSOR ;
: CLRS           6 EMIT 2 EMIT 15 EMIT 12 EMIT ; ( GREEN TEXT )
: VHTAB          3 EMIT EMIT EMIT ;
```

SCR # 43

```
( SCREEN EDITOR DISPLAY )

: SHOW-PLACE     1 4 VHTAB ." SCR# " SCR @ 2 .R
                       ."     LINE# " L# @ 2 .R
                       ."     CHAR# " C# @ 2 .R
                 21 0 VHTAB ." PAD>" 24 0 VHTAB ;
: SHOW-PAD       PAD DUP 1+ 0 22 PLOT ROT C@ SMOVE ;
: SHOW-SCREEN    CLRS  SHOW-PLACE  3 DASHES 20 DASHES
                 SCR @ BLOCK 0 4 PLOT 1024 SMOVE MARK SHOW-PAD ;
: SHOW-LINE      SCR @ BLOCK L# @ 64 * + ( LINE START ADDRESS )
                 0 L# @ 4 + PLOT 64 SMOVE  MARK ;
: +MARK          -MARK  C# @ + 63 AND C# !  L# @ + 15 AND L# !
                   MARK  SHOW-PLACE ;
: HOME        -MARK 0 C# ! 0 L# !  MARK  SHOW-PLACE ;
: NEW-LINE    -MARK 0 C# ! L# @ 1+ 15 AND L# ! MARK SHOW-PLACE ;

```

SCR # 44

```
( SCREEN EDITOR  INSERT DELETE )

: COPY-LINE      LINE PAD 1+ C/L DUP PAD C! CMOVE SHOW-PAD ;
: INSERT-LINE    DUP 1 - 14 DO I LINE I 1+ -MOVE -1 +LOOP WIPE ;
: PASTE-LINE     DUP INSERT-LINE PAD 1+ SWAP -MOVE ;
: DELETE-LINE    DUP COPY-LINE DUP 15 = IF DROP ELSE 15 SWAP
                 DO I 1+ LINE I -MOVE LOOP ENDIF 15 WIPE ;
: DELETE-CHAR    C# @ IF <SLIDE BL C/L 1 - L# @ LINE + C!
                        0 -1 +MARK UPDATE ENDIF ;
: INSERT-CHAR    C/L  C# @ - 1 > DUP  IF SLIDE> ENDIF
                           K# @ CHAR C! IF 0 1 +MARK ENDIF  UPDATE ;

: CHANGE-SCREEN     ." NEXT SCREEN NUMBER ="
                    0 BEGIN KEY DUP DUP EMIT 13 = 0=
                      WHILE 48 - SWAP 10 * +
                      REPEAT DROP SCR ! ;
```

SCR # 45

```
( NON-CURSOR KEY TABLE )

: INSERT/DELETE          K# @ CASE
    ( INSERT LINE KEY )       3 OF L# @ INSERT-LINE 1 ENDOF
    ( DELETE LINE KEY )       4 OF L# @ DELETE-LINE 1 ENDOF
    ( INSERT CHAR KEY )       5 OF L# @  PASTE-LINE 1 ENDOF
    ( ERASE LINE KEY  )      11 OF L# @   COPY-LINE 0 ENDOF
    ( DELETE CHAR KEY )     127 OF       DELETE-CHAR 0 ENDOF
    ( ANY OTHER KEY   )     INSERT-CHAR 0 SWAP
                          ENDCASE ;
;S





```

SCR # 46

```
( CURSOR KEY TABLE )

: STEP   BEGIN 33252 C@  ( STEP WHILE KEY DOWN )
         WHILE OVER OVER +MARK 100 0 DO LOOP ( TIME DELAY )
         REPEAT DROP DROP ;

: PROCESS        0 K# @ CASE
    ( CURSOR RIGHT  )    25 OF 0  1 STEP ENDOF
    ( CURSOR LEFT   )    26 OF 0 -1 STEP ENDOF
    ( CURSOR UP     )    28 OF -1 0 STEP ENDOF
    ( CURSOR DOWN   )    10 OF  1 0 STEP ENDOF
    ( HOME KEY      )     8 OF      HOME ENDOF
    ( RETURN KEY    )    13 OF  NEW-LINE ENDOF
    ( ANY OTHER KEY )    INSERT/DELETE ROT ROT DROP
                       ENDCASE
           IF SHOW-SCREEN ELSE SHOW-LINE ENDIF ;
```

SCR # 47

```
( SCREEN EDITOR MAIN LOOP )

: SEDIT    CLRS  CLEAR-PAD ." SCREEN EDITOR"
           CR CR CHANGE-SCREEN
           BEGIN HOME SHOW-SCREEN
             BEGIN
               KEY DUP DUP K# ! 27 = SWAP 12 = OR 0=
             WHILE PROCESS ( KEY NOT ESCAPE OR ERASE PAGE )
             REPEAT
             K# @ 27 = 0=
           WHILE CHANGE-SCREEN ( ERASE PAGE KEY )
           REPEAT
           HOME -MARK FLUSH ; ( ESCAPE UPDATES DISK AND EXITS )
;S


```

SCR # 6

```
( CASE, ARRAY )

: CASE   ?COMP CSP @ !CSP 4 ; IMMEDIATE

: OF     4 ?PAIRS COMPILE OVER COMPILE = COMPILE OBRANCH
         HERE 0 , COMPILE DROP 5 ; IMMEDIATE

: ENDOF     5 ?PAIRS COMPILE BRANCH HERE 0 , SWAP 2
            [COMPILE] ENDIF 4 ; IMMEDIATE

: ENDCASE   4 ?PAIRS COMPILE DROP BEGIN SP@ CSP @ = 0=
         WHILE 2 [COMPILE] ENDIF REPEAT CSP ! ; IMMEDIATE

: ARRAY   <BUILDS DUP C, * ALLOT DOES> ROT 1 - OVER C@ * + + ;
;S

```

ANIMATION *(Concluded)*

CHRIS ZERR *10932-156th Court NE*
Redmond, WA 98052
Compuserve: 71445,1240

In the last article we plotted a rectangle on the screen and moved it through a blue field at various speeds using OSTR to print db strings.. Let's try a more appropriate figure this time, constructed from plot blocks, increase the action, and poke the animaton directly into screen memory.

We will construct an 'alien' with a missile launcher, moving feet, and piercing 'eyes' that blink menacingly at us, all through the use of plot blocks poked directly into screen memory. Constructing the figure of the alien is a matter of laying out the plot blocks and identifying the code required to illuminate them in the desired way.

CONSTRUCTING THE ALIEN'S PLOT BLOCKS.

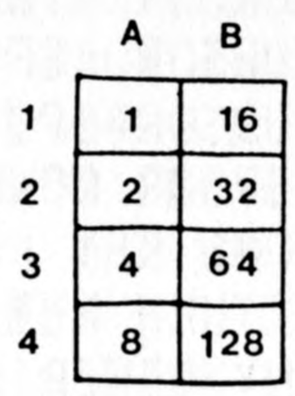
Figure 1 shows a single plot block, consisting of eight controllable portions .. A1 through A4 and B1 through B4. These eight portions coincide with the eight bits of a byte. The portions of the plot block that will be illuminated will depend on whether or not the corresponding bit in the plot byte is set. To illuminate each of the four corners of a plot block in turn, we would set our plot byte equal to each of the following numbers:

```
Upper left corner : set A1 bit (bit 0)= PLOT value 1
Lower left corner : set A4 bit (bit 3)= PLOT value 4
Upper right corner: set B1 bit (bit 4)= PLOT value 16
Lower right corner: set B4 bit (bit 7)= PLOT value 128
```

So to construct a plot block, we shade in the bit sections, add all the individual bit values to the find the total plot value, and we have it! But we are not quite finished, because plotting the plot value alone will not quite do it. The CCI code must also be assigned, and for a plot block it must have a value between 128 and 255. The effect of the CCI code will be the same, by and large, as it would be for an ASCII character attribute, but the number will be increased by 128. For example, to set the color yellow, we issue PLOT 6,3 for ASCII, but for a yellow plot block it becomes 3 + 128 or 131. (Now we know how to peek at a screen memory location to tell if there is an ASCII character there or a plot block.)[1]

You may easily experiment with plot blocks and their CCI codes in BASIC to observe the diffference various CCI codes make. Whatever character we design must be derived from a combination of plot blocks within the possibilities available to us. A crude sketch on graph paper is the easiest way, possibly, and permits an instant translation into the correct plot codes.

Fig 1.



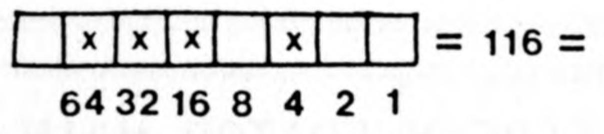
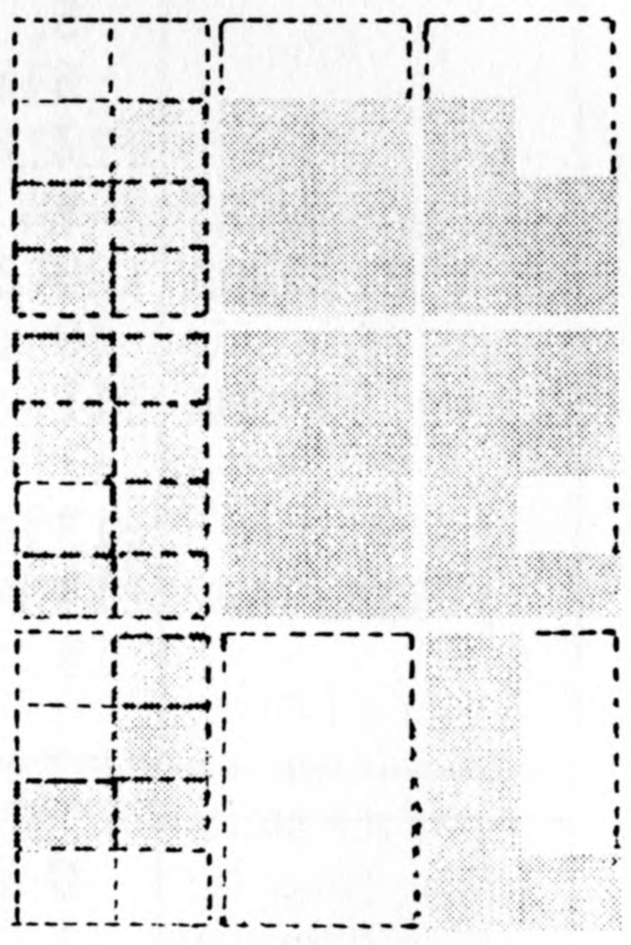


Fig 2.



Our alien has two different forms in this program. In the first form his left foot is raised; in the other his right foot is raised. Figures 2 and 3 illustrate the composition of these two figures. The outline of the plot blocks has been exaggerated in Fig 2 so you can see how the plot codes for this figure were derived. The upper left plot block has bits 2, 3, 5, 6, and 7 set, giving a value of 4 + 8 + 32 + 64 + 128 or 236 for this plot block. You should try to derive the plot codes for the remaining blocks as an exercise. The lower left block is shown with the proper bits set for you in Figure 2.

The plot blocks from Figure 2 contain the values 236, 238, 206, 251, 191, 116, 0, 143. The zero (0) count is necessary to act as a filler which says a null block is to be plotted in that location. In actual practice, each of these plot blocks must also have a CCI code following it, so a completed command byte string for Figure 2 would be 236, 131, 238, 131, 206, 131, etc.

We may mix plot blocks with ASCII characters in the plot command string, and our alien requires this to plot his 'eyes', which are quotes in ASCII. You will find them in the plot string from this Basic example, which plots the alien in one of his two forms on the screen:

```
100  X=30780 : REM INITIAL SCREEN MEMORY LOCATION
110  FOR A=1 TO 3 : REM PAINT 3 VERTICAL PAIRS OF BLOCKS
120  FOR B=1 TO 6 : REM THREE PAIRS OF BYTES FOR X AXIS
130  READ Z : REM GET POKING DATA
140  POKE X,Z : REM POKE ALT PLOT CODES & CCI
150  X=X+1
160  NEXT B
170  X=X+122 : REM PLOTS DOWN ONE BLOCK ON Y AXIS
180  NEXT A : REM PLOT NEXT VERTICAL PAIR
190  END
200  DATA 236,131,238,131,206,131,251,131,34,25,191,131
210  REM '34,25' IS ASCII ["] IN RED
220  DATA 116,131,0,0,143,131 : REM '0,0' IS FILLER PAIR
```

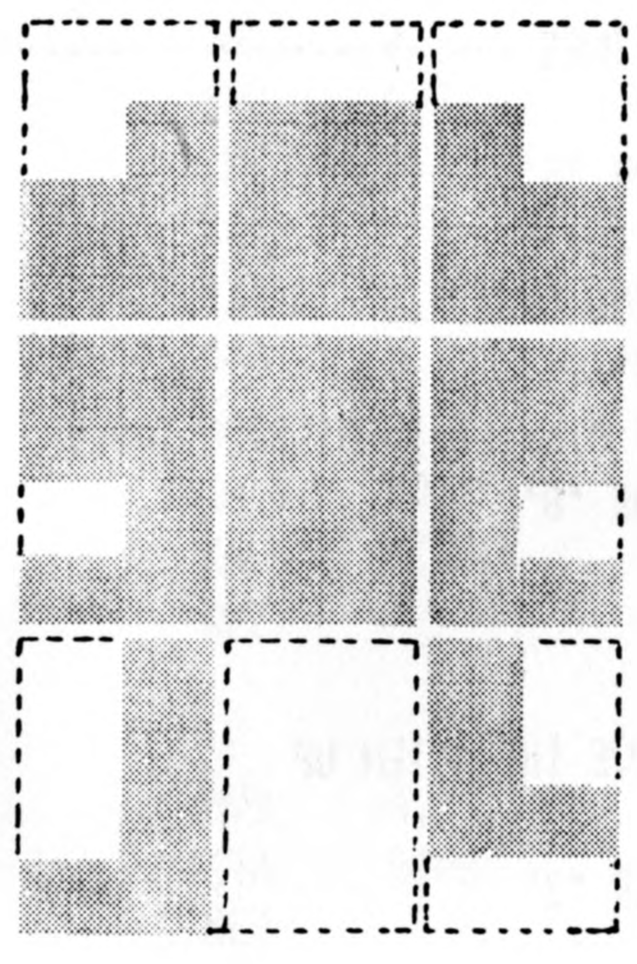


Fig 3.

This program may also be written as one line:

```
10 PLOT 3,30,10,19,2,254,236,238,206,255,3,30,11,2,254,251,
        255,6,25,34,6,3,2,254,191,255,3,30,12,116,0,143,255
```

All we have done so far is paint one of the two alien forms. The numbers for the other form, including the 'eyes', and the CCI code are:

```
236,131,238,131,206,131,251,131,
34,27,191,131,248,131,0,0,71,131
```
(See Figure 3.)

The principle is very simple. We now have a subject ready for use.

BORDERS.

Every game has a border, or a boundary, but it isn't always a visible one. For example, the edges of the CRT are always a finite boundary for any plotting. The programmer needs a border, visible or not, so he can check to see if his animaton has gone as far in one direction as it can go. If such a check were not possible, animation would stop when the CRT boundary were reached, or an animaton might leave the screen when this was not desired. If you look at an invisable boundary (in a properly programmed routine) you will probably see an ASCII character from 1 to 255, but a CCI code of zero (0), so it doesn't show. Whatever the choice of boundary encoding, it must be selected so it is unique to the boundary, and not used anywhere else on the screen. It is entirely possible to make each of the four boundary codes different so the program knows exactly which boundary it has reached. This is much more easily done if the actual boundary is invisible, for otherwise the encoding changes would have to be visible sooner or later (if the CCI code were not zero, for example.)

Our program uses a border made of the hatch character (96 decimal) and a CCI code of 38 (blue on white). We are not going to let the alien move off the screen, so our border encoding will be a warning to us that a boundary is present, and that the direction of the next move must be changed.

The border is poked directly into screen memory by the subroutines TANDB and SIDES, beginning at memory address 7000H. The blocks representing the animaton are also poked directly into screen memory. The inital location of these blocks is established in the nineth line of START, where location 30790 is recorded in LOC. You may change this location, provided you replace it with a valid screen memory address.

MOVEMENT

We will use keyboard scanning to let the operator move the alien in one of four directions and to fire the alien's missile.[2] The scanning procedure makes it unnecessary for the program to wait for a key press before proceeding with the automation; otherwise, the left foot might just stay 'put' until we pressed a key. The technique we use was describ ed by Steve Perrigo in Colorcue, June/July, 1982. The following assignments have been made to direct alien motion:

''8'' = move up, ''2'' = move down, ''4'' = move left, and ''6'' = move right.

Entering a ''5'' will 'fire' the missile. This pattern coincides with most joystick wiring. If your joystick uses the arrow keys, these may be substituted for the numbers readily. Notice that diagonal movement is possible in our routine by pressing two keys at the same time (or moving the joystick between two absolute directions.)

FIRING THE MISSILE

When the ''5'' key is pressed, the routine FIREX is called. It checks first to see if a missile is already being fired. If so, the routine is aborted; otherwise FIREX finds the alien's current position and direction of movement. We are choosing to fire the missile in the direction of alien travel, and so we must plot a course for the missile. To make things 'neat' we fire the missile from the center of the alien. You will notice that the MISSLE routine is called several times for each single movement of the alien. This is necessary to prevent the alien from moving right along with the missile. The missile will travel until it hits the border.

The code in Listing I is well documented, and, as with all assembly code, it needs to be read to be understood. You might try experimenting with the time delay at label MOVEIT. You might also try moving the alien on a totally blue background (as with the rectangle, last time.) This kind of experimenting can be very inspirational when designing a game. As an additional exercise, find a way to remove the call to OSTR, and make this program completely FCS-version independent.

We also know, now, how to expand the program developed in the last last article, making use of a border sensor, and introducing a method by which the rectangle 'knows' when it is about to tred onto the blue field, and when it is retracing a previous path. You may reprogram that listing to guide the rectangle through its first complete connecting loop, after which it will maneuver itself without straying from the initial path. □

[1] The CCI codes are discussed thoroughly in the CCII Programmer's Manual. Also see 'Color Graphics' by David Suits.

[2] Note: This routine will not operate properly on a 3651. To duplicate the function, use the GTCHA routine and CHRINT setup from David Suit's input routine in Colorcue. Write to Colorcue for details if you need some help.

```
        ;LISTING I.
;
;*****  A GAME SIMULATION  by Chris Zerr. 6/30/84  *****

        OSTR    EQU     182AH   ;For v8.79,9.80
                                ;See ROM tables for 6.78

        ORG     9000H

;------------------- SETUP PROCEDURES -------------------

START:  XRA     A               ;SET A=0
        OUT     8               ;DISABLE KEYBOARD
        STA     DIREC           ;CLEAR DIRECTION
        STA     SWITCH          ; AND SWITCH
        LXI     H,0             ;MISSILE LOCATION
        SHLD    FLOC            ; SET TO ZERO
        LXI     H,CLEAR         ;CLEAR SCREEN
        CALL    OSTR

                                ;INITIAL POSITION FOR ALIEN
        LXI     H,30780         ; START SOMEWHERE IN THE
        SHLD    LOC             ; MIDDLE OF THE SCREEN

        LXI     H,7000H         ;LETS MAKE A BORDER
        CALL    TANDB           ;TOP & BOTTOM - DO TOP
        LXI     H,7F80H
        CALL    TANDB           ;DO BOTTOM
        LXI     H,7000H
        CALL    SIDES           ;DO LEFT SIDE
        LXI     H,707EH
        CALL    SIDES           ;DO RIGHT SIDE
;---------------------- MAIN PROGRAM ----------------------

REPETE: LHLD    LOC             ;LOAD HL WITH ALIEN LOCATION
        LDA     SWITCH          ;GET CHANGE SWITCH
        ORA     A
        JZ      CLOSE           ;IF ZERO, USE LEFT FOOT DOWN
        LXI     D,ALIENR        ; ELSE ALIEN RIGHT
        XRA     A               ; FOOT DOWN
        STA     SWITCH
        JMP     MOVEIT

CLOSE:  LXI     D,ALIENL        ;ALIEN LEFT FOOT DOWN
        MVI     A,255           ;SET THE FLAG SO WE CAN
        STA     SWITCH          ; ROTATE THE ALIEN MOVEMENT

MOVEIT: CALL    MOVE            ;MOVE THE ALIEN
        MVI     C,70            ;DELAY ABOUT 1.5 SECONDS
AA:     MVI     B,255           ;  ( DELAY ROUTINE )
LL:     DCR     B               ;
        JNZ     LL              ;
        DCR     C               ;
        JNZ     AA              ;
        CALL    KEY             ;CHECK KEYBOARD INPUT
        JMP     REPETE          ; AND DO IT AGAIN AND AGAIN!

; ------------------- END OF MAIN PROGRAM -------------------
```

```
;---------------------- SUBROUTINES ----------------------

KEY:    MVI     A,10            ;CHECK FOR "5"
        OUT     7               ;SEND IT OUT
        IN      1               ;GET STATUS
        CPI     254             ;IS IT 5??
        CZ      FIREX           ;YES..PUT MISSILE IN MISSILE TABLE
        MVI     A,7             ;CHECK FOR "8"
        OUT     7
        IN      1
        CPI     254
        CZ      UP              ;YES...MOVE THE ALIEN UP
        CALL    MISSLE
        MVI     A,13            ;CHECK FOR "2"
        OUT     7
        IN      1
        CPI     254
        CZ      DOWN            ;YES...MOVE THE ALIEN DOWN
        MVI     A,11            ;CHECK FOR "4"
        OUT     7
        IN      1
        CPI     254
        CZ      LEFT            ;YES...MOVE THE ALIEN LEFT
        MVI     A,9             ;CHECK FOR "6"
        OUT     7
        IN      1
        CPI     254
        CZ      RIGHT           ;YES...MOVE THE ALIEN RIGHT
        CALL    MISSLE          ;IF THERE IS A MISSILE TO MOVE,
                                ; MOVE IT AGAIN...
        RET

UP:     LHLD    LOC                     ;PROCESS ALIEN TO GO UP
        LXI     D,ERASE                 ; BUT FIRST ERASE
        CALL    MOVE                    ; THE OLD ALIEN
        LHLD    LOC
        LXI     D,0FF80H                ;SUBTRACT 128 FROM HL REG
        DAD     D
        MOV     A,M                     ;IF WE ARE AT THE BORDER
        CPI     96
        RZ                              ; WE DON'T ADVANCE THE ALIEN
        SHLD    LOC
        CALL    MCHECK                  ;CHECK TO SEE IF A MISSILE
                                        ; IS TRAVELLING-
        RNZ                             ; IT IS SO FORGET IT!
        MVI     A,1                     ; IT ISN'T SO-
        STA     DIREC
        RET

DOWN:   LXI     D,ERASE                 ; SO, YOU WANT TO MOVE
        LHLD    LOC                     ;  DOWN!?  WELL FIRST
        CALL    MOVE                    ;  ERASE THE OLD ALIEN
        LHLD    LOC
        LXI     D,128                   ;GET OUR DOWNWARD OFFSET
        DAD     D                       ; WE MUST CHECK
        DAD     D                       ; ONE BELOW THE
        DAD     D               ; BOTTOM OF THE
        MOV     A,M             ; ALIEN -
        CPI     96              ; TO SEE IF WE ARE AT
        RZ                      ; THE BORDER -
        LXI     D,0FF80H        ;WE'RE NOT!
        DAD     D               ;SINCE WE ONLY NEED TO MOVE
        DAD     D               ; ONE 128 BYTES, WE MUST
        SHLD    LOC             ; REMOVE THE EXTRAS FROM ABOVE
        CALL    MCHECK          ;MOVE A MISSILE (IF ANY)
        RNZ                     ;IF A MISSILE IS IN PROGRESS
                                ; DON'T BOTHER WITH THIS NEW
                                ; LOCATION
        MVI     A,2             ; OUR DIRECTION IS 2 FOR DOWN
        STA     DIREC
        RET

LEFT:   LXI     D,ERASE         ;AGAIN ERASE THE OLD LOCATION
        LHLD    LOC
        CALL    MOVE            ;DO THE ERASE
        LHLD    LOC             ;REGAIN OUR CURRENT LOCATION
        DCX     H               ;DOWN BY ONE
        MOV     A,M             ;THIS IS OUR CCI COLOR CODE
        CPI     38              ;IF WE ARE HERE, WE WILL NOT
        RZ                      ; MOVE
        DCX     H               ;BECAUSE WE CAN MOVE
        SHLD    LOC             ; BUMP IT DOWN AGAIN AND SAVE
                                ; THIS PUTS US ON AN EVEN X,Y
                                ; OR "CHARACTER,CCI" CODE
        CALL    MCHECK          ;CHECK FOR MISSILE IN PROGRESS
        RNZ                     ;YES..SO RETURN TO THE CALLER
        MVI     A,3             ;OUR DIRECTION IS 3
        STA     DIREC           ; WHICH IS LEFT
        RET

RIGHT:  LXI     D,ERASE
        LHLD    LOC
        CALL    MOVE
        LHLD    LOC             ;GET OUR LOCATION
        LXI     D,0006          ;JUST 6 BYTES TO THE RIGHT
        DAD     D               ; IS ALL WE WANT..NOT THE WORLD
        MOV     A,M             ;DID WE HIT THE BORDER?
        CPI     96
        RZ                      ;SORRY CHARLIE..ONLY THE BEST
        LXI     D,0FFFCH        ;NOW BACK UP 4 BYTES
        DAD     D               ;DO IT
        SHLD    LOC             ;SAVE OUR NEW HOME
        CALL    MCHECK
        RNZ
        MVI     A,4             ;OUR DIRECTION IS 4
        STA     DIREC           ; WHICH IS RIGHT...RIGHT?
        RET

FIREX:  CALL    MCHECK          ;IF A MISSILE IS STILL IN
        RNZ                     ; PROGRESS..LEAVE
        LDA     DIREC           ;GET ITS DIRECTION
        ORA     A               ; IF ZERO
        RZ                      ; THERE IS NOWHERE TO GO
        LHLD    LOC             ;THE FOLLOWING WILL
        CPI     1               ; POSITION THE HL REG
        JZ      POSUP           ; TO POINT AT A LOCATION
        CPI     2               ; IN WHICH THE MISSILE
        JZ      POSDN           ; WILL LEAVE THE ALIEN
        CPI     3
        JZ      POSLF
        CPI     4
        JZ      POSRT
        RET

;)))---> HOW THE FOLLOWING ROUTINE WORKS
;
;       WHAT I HAVE CHOSEN IS TO LAUNCH A MISSILE
; FROM THE CENTER OF THE ALIEN IN WHICH THE DIRECTION
; OF THE ALIEN IS OR WAS PRESENTLY MOVING.  SO FOR OUR
; FIRST POSITION, POSUP, WE GET OUR CURRENT LOCATION,
; THEN ADD 2 BYTES TO IT.  THIS WILL THEN BE THE FIRST
; LOCATION OF THE LAUNCHED MISSILE.

POSUP:  INX     H               ;POSSIBLE LOCATIONS ARE:
        INX     H
        JMP     SAVPOS
                                ;        ^
POSDN:  LXI     D,0258          ;      PCPCPC
        DAD     D               ;    < PCPCPC >
        JMP     SAVPOS          ;      PCPCPC
                                ;        V
POSLF:  LXI     D,0128
        DAD     D               ;HL IS ALWAYS POINTING
        JMP     SAVPOS          ; TO THE FIRST POSITION
                                ; OF THE ALIEN
POSRT:  LXI     D,0132
        DAD     D

SAVPOS: SHLD    FLOC            ;SAVE THE POSITION
        RET

MISSLE: LHLD    FLOC            ;PLOT THE MISSILE
        MOV     A,H             ;IF HL ZERO,
        ORA     L               ; THEN THERE IS NO
        RZ                      ;  MISSILE TO PLOT
        LDA     DIREC
        CPI     1
        JZ      FIREUP
        CPI     2
        JZ      FIREDN
        CPI     3
        JZ      FIRELT
        CPI     4
        JZ      FIRERT
        RET
```

```
FIREUP: CALL    MERASE          ;ERASE THE OLD MISSILE
        DCX     H               ;HL RETURNED UP'ED BY ONE
        LXI     D,0FF80H        ;SUBTRACT 128 BYTES
        DAD     D
        SHLD    FLOC            ;SAVE IT
        MOV     A,M
        CPI     96
        JZ      NOMORE          ;CAN'T PASS THE BORDER
        MVI     M,110           ;PLOT THE NEW MISSILE
        INX     H
        MVI     M,1             ;USE COLOR OF RED
        RET

FIREDN: CALL    MERASE          ;ERASE THE OLD MISSILE
        DCX     H
        LXI     D,0128          ; TO PLOT IT DOWN 1
        DAD     D               ; ADD 128 BYTES
        SHLD    FLOC
        MOV     A,M
        CPI     96              ;WE ARE AT THE BORDER
        JZ      NOMORE          ; SO RETURN
        MVI     M,110
        INX     H
        MVI     M,2             ;USE COLOR GREEN
        RET

FIRELT: CALL    MERASE          ;ERASE OLD MISSILE
        DCX     H               ;TO GO LEFT
        DCX     H               ; WE MUST SUBTRACT 3
        DCX     H               ; BECAUSE MERASE ADDS ONE
        MOV     A,M
        CPI     96
        JZ      NOMORE          ;CAN'T GO THROUGH THE BORDER
        SHLD    FLOC
        MVI     M,105           ;LOWER CASE I
        INX     H
        MVI     M,5             ;COLOR MAGENTA
        RET

FIRERT: CALL    MERASE          ;ERASE
        INX     H               ;INCREASE HL BY ONE
        MOV     A,M
        CPI     96
        JZ      NOMORE          ;SORRY PAL
        SHLD    FLOC            ;IT'S GOOD! SAVE IT
        MVI     M,105
        INX     H
        MVI     M,7             ;COLOR WHITE
        RET

NOMORE: LXI     H,0             ;MISSILE NO MORE, SO
        SHLD    FLOC            ;CLEAR IT FROM THE
        RET                     ; MISSILE TABLE
```

```
MCHECK: LHLD    FLOC            ;CHECK IF MISSILE IS
        MOV     A,H             ;IN PROGRESS
        ORA     L
        RET

MERASE: MVI     M,' '           ;ERASE THE OLD MISSILE
        INX     H
        MVI     M,0
        RET

MOVE:   MVI     B,3             ;PLOT THE ALIEN ON THE SCREEN
ROW:    MVI     C,6             ;THIS ROUTINE DOES THE ACTUAL
COLUMN: LDAX    D               ;PLOTTING OF THE ALIEN.  H&L
        MOV     M,A             ; ARE PASSED TO THIS ROUTINE
        INX     D               ; AND THE ALIEN IS THEREFORE
        INX     H               ; PLOTTED.
        DCR     C
        JNZ     COLUMN
        PUSH    B
        LXI     B,0122          ;AT THE END OF 6 WE MUST
        DAD     B               ; POSITION OURSELVES AT THE
        POP     B               ; NEXT LINE.
        DCR     B               ;EXAMPLE:
        JNZ     ROW
        RET                     ;HL=7000H
                                ; AT THE END OF 6 COUNTS,
                                ; HL=7006H, SO WE ADD 7AH(122)
                                ; HL=7080H, WHICH IS RIGHT
                                ; BELOW WHERE HL WAS (7000H)

;PAINT BORDER---------------------------------------------

TANDB:  MVI     C,64            ;TOP AND BOTTOM OF BORDER
        MVI     M,96
        INX     H
        MVI     M,38
        INX     H
        DCR     C
        JNZ     TANDB+2
        RET

SIDES:  LXI     D,128           ;SIDES OF BORDER
        MVI     C,31
SIDE1:  MVI     M,96
        INX     H
        MVI     M,38
        DCX     H
        DAD     D
        DCR     C
        JNZ     SIDE1
        RET
```

```
;------------------------ DATA STORAGE ----------------------

SWITCH: DS      1           ;ALIEN SWITCHING
DIREC:  DS      1           ;MISSILE DIRECTION
FLOC:   DS      2           ;MISSILE LOCATION
LOC:    DS      2           ;ALIEN  LOCATION

CLEAR:  DB      6,0,15,27,24,12,3,64,0,239

;       ALIEN WITH LEFT FOOT DOWN

ALIENL: DB      236,131,238,131,206,131
        DB      251,131,34,25,191,131
        DB      116,131,0,0,143,131

;       ALIEN WITH RIGHT FOOT DOWN

ALIENR: DB      236,131,238,131,206,131
        DB      251,131,34,28,191,131
        DB      248,131,0,0,71,131

;       DATA TO ERASE THE ALIEN

ERASE:  DB      32,0,32,0,32,0
        DB      32,0,32,0,32,0
        DB      32,0,32,0,32,0

        END     START
```

Colorcue will be saying 'goodby' ......

The next issue of Colorcue, Vol VI, No 6, will be the last for this seven year old publication. Diminishing support is the primary reason, along with greatly increased publication costs. We might have simply economized on our formatting and means of distribution, but it has been clear that what is most needed is a consolidation of effort among the various CCII organizations. With consolidation, we have an opportunity to maintain our activity and still provide the necessary publication services to subscribers.

The Rochester User Group is the most appropriate center for this consolidation. Its publication, DATA CHIP, and its fine user library provide subscribers with a wealth of continuing materials for the Intecolor computers.

Beginning early in 1985, CHIP will be expanded to include materials normally published in Colorcue. Authors will continue to write for publication in CHIP. With this added contribution, CHIP can be a more frequent and expansive publication, which it well deserves to be.

The price of membership in the Rochester User Group has been increased from $10 to $15 per year ($20.00 USA for overseas) to cover the expanded publication of CHIP. I urge you to lend your total support to this effort. Send your check for $15.00 today to

GENE BAILEY: 28 Dogwood Glen, Rochester, NY 14625

Your prompt response is necessary for good planning in Rochester, and to be sure you don't forget and therefore miss some of the exciting material ready for release (including a new article by Ben Barlow on expanding the disk capacity of the CCII!) Send materials for publication in CHIP to Rick Taubold, 197 Hollybrook Road, Rochester, NY 14623.

The last issue of Colorcue will contain a complete index of the major CCII publications, and a fine set of articles recently submitted for publication. It will, no doubt, be a large issue, and will also, with certainty, be late. There is a large amount of midnight oil to be burned before it is ready. Meanwhile, I hope we will see our first issues of the expanded CHIP newletter. As always, these are your publications, and your active support is required to make them valuable to CCII users. The end is not yet in sight, friends!

A reminder, too, that Australia has been very active lately in the CCII department. You will enjoy a subscription to CUVIC. See Colorcue, Vol VI, No 3, page 31 for membership information.

COLORWORD *A re-review*

DOUG GRANT

Doug Grant, the librarian of CUWEST in Australia, sent a very enthusiastic letter to David Ricketts, the author of the GEMINI printer review in the last issue. His subject was COLORWORD, the word processor distributed by PPI in Australia. Doug feels the COLORCUE review was less laudatory than it might have been and points out specific features of the software that have proven valuable to him.

We are pleased to excerpt it here, with permission, and print a few examples of type from Doug's Epson printer.

"(Dear David,)..We are about even in our standard of Tightwadishness. I took a good look at Comp-U-Writer long ago and decided that I didn't want a WP which couldn't use all the codes of the Epson printer. I purchased Wordthis and Wordthat and Thisaword and Thataword and probably got to around Comp-U-Writer's price in total anyway. Now, we have a very good programmer in our group, Chris Teo who came up with COLORWORD, and if you don't send off the $50 to PPI as soon as you can drag out your chequebook then you just don't deserve to own a printer as good as the Gemini appears.

"(The enclosed page is a printout of a disk file I use to demonstrate COLORWORD's use of printer control codes) and you will see that with expert use...you can really get a message through. All of these codes are added to your text following your pressing INSERT CHAR, of course, and this includes CTL @ and CTL G. ESC appears as '[', in blue, and CTL @ and CTL G as @ and G in blue. You will find out for yourself about the 'A7' and etc.

"Have a good look at the review in Colorcue of May/June 1984 but don't let it deter you. We have members who have both Comp-U-Writer and COLORWORD v4.5 who rarely use C-U-W except when they require two column work or lots of text centering. COLORWORD misses these two features, but Chris will probably come up with that later. The only thing you may get a bit excited about is that when you go back in the text a little way to add a few more words here and there, the word wrap still does its job but sometimes moves the remainder of the line down a line, and if you move the cursor past that line you may think you have lost a line or so. No worries though, you simply go back to the beginning of the paragraph and hit CMD R to reform the paragraph. Even if you didn't do this, the printout would still be correct, as you have to hit RETURN twice to force new lines anywhere and you can't lose any text without really trying.

"The underline is a true underline and not a series of dashes and in editing you can use the colour pad to whip through by paragraphs or lines in either direction. There is no 'undelete' so you must make fewer errors in this regard. If you accidentally hit AUTO you go off into a 'search' mode but don't panic, just hit AUTO a couple more times and you are back to the text again.

"You can set up your output baud rate, double spacing, spaced print, etc. at each sitting, but there is a provision for you to make up your own text file incorporating all the parameters you wish, and load it up when you start work. I have 24 good sized letters on one side of the disk (initialized to 07) which I have in the drive at this moment, and I can exit the text and delete any of them if I run out of disk space; or put another disk in. At the top of the screen I see that I still have 12701 characters of space left in this text if I want to go on all night at my rate of typing, but I won't. Oh well; back to the cot for another read of COLORCUE.

"All the best to you from,
Doug Grant."

INTECOLOR BULLETIN BOARD FOR CCII

Steve Reddoch's proposal for a bulletin board for the CCII has not had any response from readers. The Santa Barbara subscriber has prepared a program for free distribution to CCII owners in machine language to service both 300 and 1200 baud. Our two hundred users are not enough to support such a project, it seems, and there are many of us who can sympathize with Steve's disappointment that this project has not seen the light of day. But there has been progress! Intecolor Corporation has recently announced the establishment of a Bulletin Board for all its products, including the CCII, 3600 and 8000 computers. It is located in the Northwest USA. The telephone number is (206) 4833460, and the Sysop ("System Operator") is Bob Morgan, an Intecolor dealer. While it is a CP/M board, TERMII software should work well. We encourage you to take advantage of this new opportunity. The board will also include news on Intecolor's DataVue computers. These are being offered at a special price to CCII owners. They are CP/M machines, with built-in hard disks and full CP/M features. The computer is available without a terminal, and the CCII may be used as a terminal with it. Software is available for the terminal conversion from Rick Taubold and Tom Devlin, which they developed with Myron Steffy for interfacing with the Morrow computer line.

KEYBOARD UPGRADES

It's not too late to upgrade your CCII keyboard to the full 117 keys. Howard Rosen (PO Box 434, Huntington Valley, PA 19006) has enlarged keyboard covers available for $20.00 and switches for $2.00 each. Key caps may be ordered from Arkay Engraving (see last issue). Intelligent Systems Corporation in Huntsville has some of the same materials. Give them a call at (205) 881-3800 or write 12117 Comanche Trail, Huntsville, AL 35803.

John Ker in Los Angeles is a network subscriber. His Compuserve network address is 71735, 1673, and his Source address is TCP733. Welcome aboard, John! I've enjoyed our modem talks.

Control CODES for **COLORWORD** and EPSON with GRAFTRAX PLUS ROM Set.

ESC 4	turns on *Italics*
ESC 5	turns off Italics
A7 on	turns on **enlarged** characters
BLUE KEY	turns off enlarged characters
BL/A7 OFF	turns on condensed characters
GREEN KEY	turns off condensed characters
ESC E	turns on **emphasized** characters
ESC F	turns off emphasized characters
ESC G	turns on **double** character
ESC H	turns off double character
ESC S	turns on Subscript mode
ESC S followed by CONTROL @	turns on Superscript mode
ESC H	turns off Subscript or Superscript mode.
ESC 0	gives 8 lines per inch
ESC 2	gives 6 lines per inch
ESC -	turns on <u>Underline</u>
ESC - followed by CONTROL @	turns off Underline
Control G	Sounds the Printer Bell.
ESC @	Initializes the Printer and resets the top of form position.

COMBINATIONS OF <u>VARIOUS</u> CONTROL CODES
GIVE **VARIOUS** RESULTS

tiny — PASCAL (conclusion)

This final part of the series is dedicated to the 'bread-n-butter' Tiny-Pascal commands. We can not hope to give a textbook treatment of these commands, so we will restrict ourselves to some typical examples of their use. Your familiarity with other programming languages, and the use of a good Pascal tutor will enable you to understand and apply our contents on the CCII.

A NEW SCREEN EDITOR

A new tool is now available to help in the exploration and implementation of Tiny-Pascal on the Compucolor in the form of a full screen editor, written by Bill Greene, and housed in the Chip Users Group Library. This editor significantly reduces the task of entering and editing both Forth and Tiny-Pascal programs on Forth screens. The screen editor, along with two Forth disassemblers and an 8080 assembler are contained in screens on Chip Disk #121. This disk complements Bill Greene's implimentation of Forth, called Forth8, contained on Chip Disk #120. The editor, and any of the other features, may be compiled into the Forth8 core program and saved on disk as an augmented version of Forth (call it FORTHE.) The screen editor is not compatible with Jim Minor's version of Forth on Chip Disk #46. However, the editor can be used to enter and edit Tiny-Pascal screens. On the other hand (all confusion aside, folks), Jim Minor's Tiny-Pascal compiler, which resides in screens in Chip Disk #83, can be compiled into Greene's Forth8 and saved as an augmented version of Forth8 (and call it FORPAS.) It is unfortunate that both the screen editor and the Tiny-Pascal compiler can not be compiled into the same Forth version, due to memory requirements, but using them in two separately augmented versions each will facilitate Tiny-Pascal operations. a) Use FORTHE to enter, edit and save the programs, and b) use FORPAS to compile and run the programs.

If this seems too much, use the line editor already described previously in these articles, or you can send two disks, and $2.00 for return postage, to the author at the above address for these new materials and instructions. Be sure to specify your FCS version, and add $10.00 to have Chip supply the disks. The four sides you receive will contain Chip Disk #120, #121, FORTHE/FORPAS, described above, and all the program examples in the Tiny-Pascal series. Back up the disks when you receive them. [You do realize what a bargain this is! Two languages for the price of none. ED]

TINY PASCAL COMMANDS

It is called 'Tiny-Pascal' because it contains a subset of the fully-implemented version of the language. This subset is sufficiently complete to allow for effective programming in many instances. When viewing Tiny-Pascal you will find many commands similar to those of the more common Basic language dialects, such as IF..THEN..ELSE and FOR..TO..DO. Other commands are found only in very extended Basic dialects, like BEGIN..END, WHILE..DO, REPEAT..UNTIL, CASE..OF, and the PROC and FUNC structures.

You will learn best by working the examples at the computer. Most examples have been limited to a single screen to conserve time for entry and editing. You must carefully observe the punctuation and program structure as it is shown in the listings. Review Part 2 of this series in Colorcue, Vol VI, No 3 to remind yourself of the commands already covered, like WRITE, READ, and NEWLINE.

This version of Tiny-Pascal is intimately intertwined with the Forth language as well, as we have seen, which is used to facilitate program execution. Defining Forth commands in such a way that they may be injected into Tiny-Pascal makes the programs significantly more versatile and useful. In mastering Tiny-Pascal, there is great incentive to master Forth, one language helping to clarify the other. We will demonstrate this in some examples.

The Pascal statement format is very important. Pascal commands are used in both single and compound statements in the listings. Single statements use a command word followed by a semicolon (;). A compound statement begins with a BEGIN and ends with an END and semicolon, surrounding multiple command words and their arguments. (If a compound statement is the last statement before an END, the trailing semicolon may be omitted. This is the only place where the semicolon is optional.) These compound statements form a functional sector, much like a subroutine, and which holds the great strength of a structured language like Pascal. Such a structure makes reading programs, without line numbers, fluent and logical.

To combine the functional sectors into a unified program we group them into a 'block'. The block is terminated with a period (.) to tell the assembler where the program ends. You will see in the examples that compound statements may be nested within a block.

We use two types of comment delimiters in Tiny-Pascal, ( ) for Forth and // for Pascal. Our programs always begin in the Forth domain, so the parentheses will be seen there. Following the Forth word PASCAL, we are in a Pascal environment, and the slash must be used, since parentheses are not defined as comment delimiters in Tiny-Pascal. Note that a space between the delimiter and the comment is required in Forth, but not in Pascal. Finally, following the Tiny-Pascal END statement, the computer is returned to the Forth environment, and rules for Forth will again prevail.

Each of the sample programs explores a different major statement function. The WHILE..DO and REPEAT..UNTIL statements are shown in Listings 1 and 2 respectively. Note that the WHILE and UNTIL are followed by a conditional expression (telling 'while' and 'until' what?) which is either true or false. (Conditional expressions are familiar to Basic programmers from their use in the IF..THEN format.) Examples include 'X = Y', 'U greater.than V', and 'Z not.equal A.' Note also that no colon is used in a conditional expression before the 'equals' sign as it is in the assign statement. In the WHILE..DO statement, a 'true' condi-

DOUG VAN PUTTE *18 Cross Bow Drive, Rochester, NY 14624*

tion will cause the statement following the DO (whether single or compound) to be repeated until the conditional expression becomes 'false.' The 'true' condition invokes a looping effect, similar to the FOR..NEXT statement in Basic. When the conditional expression becomes 'false', program control will be advanced to the point following the semicolon (;) just after the statement associated with the DO. The final ';S' in this and other listings is a FORTH command (remember we are back in the FORTH environemt after the END. appears!) This FORTH symbol has various meanings in different circumstances. Here it means simply 'stop execution of screen and go back where you came from.'

Although similar to WHILE..DO, the REPEAT..UNTIL statement has one basic difference; for a given conditional expression, the statements following REPEAT will be executed one time more (than in WHILE..DO) because WHILE..DO tests the conditional before executing the statements, and REPEAT..UNTIL tests the condition after executing the statements. Following a 'false' condition, control is passed to the first statement following the first semicolon to appear after the UNTIL.

Experiment with the programs of Listings 1 and 2 by first loading the FORTH editor. Then, using a disk with blank screens,use the editor to type and save the program on a screen of your choice. Now, load the Tiny-Pascal compiler from its disk and run it. Reload the screen disk. Type the program screen number followed by 'LOAD' and a carriage return. This will compile the program. If the compiler finds an error, the screen editor must be reloaded and the code corrected. When the code has finally compiled correctly, it may be 'run' by just typing the program name (such as 'WHILEEXAMPLE'). Follow similarly for all the other listings.

```
SCR # 15
  0 ( *LISTING 1: WHILE DO EXAMPLE ) PASCAL DECIMAL
  1 PROGRAM WHILEEXAMPLE;
  2 VAR
  3    X, Y, SQUARE : INTEGER;
  4 BEGIN
  5    NEWLINE; WRITE ( 'ENTER NO. TO SQUARE ' );
  6    READ ( #Y );
  7    X := 1;
  8    SQUARE := 0;
  9    WHILE X <= Y DO
 10      BEGIN
 11        SQUARE := SQUARE + Y;
 12        X := X + 1
 13      END;
 14    NEWLINE; WRITE ( #Y, ' SQUARED IS ' , #SQUARE );
 15 END. ;S
```

In contrast to the looping statement types we have just discussed, the IF..THEN..ELSE statement, demonstrated in Listing 3, is designed for branching, just as in BASIC, and the statements which follow are executed only once. In this statement format, a 'true' conditional will cause action by the statements assigned to THEN, passing next beyond any statements allied with the ELSE. Otherwise action will

```
SCR # 16
  0 ( *LISTING 2: REPEAT UNTIL EXAMPLE ) PASCAL DECIMAL
  1 PROGRAM UNTILEXAMPLE;
  2 VAR
  3    X, Y, SQUARE : INTEGER;
  4 BEGIN
  5    NEWLINE; WRITE ( 'ENTER NO. TO SQUARE ' );
  6    READ ( #Y );
  7    X := 1; SQUARE := 0;
  8    REPEAT
  9      BEGIN
 10        SQUARE := SQUARE + Y;
 11        X := X + 1
 12      END;
 13    UNTIL X > Y;
 14    NEWLINE; WRITE ( #Y, ' SQUARED IS ' , #SQUARE );
 15 END. ;S
```

```
SCR # 17
  0 ( *LISTING 3: IF THEN ELSE EXAMPLE ) PASCAL DECIMAL
  1 PROGRAM IFTHENELSEEXAMPLE;
  2 CONST
  3    X = 7;
  4 VAR
  5    Y : INTEGER;
  6 BEGIN
  7    NEWLINE; WRITE ('GUESS A NUMBER (1 TO 10)' );
  8    READ ( #Y );
  9    NEWLINE;
 10    IF (Y <> X)
 11      THEN
 12        WRITE( 'SORRY, WRONG NUMBER ' )
 13      ELSE
 14        WRITE ( 'YES, THE NUMBER IS ' , #X );
 15 END. ;S
```

derive from the statements associated with ELSE, then continue onward. The ELSE portion is optional, which means it may be omitted. In this instance, the IF..THEN rules apply just the same as before (just as they do in BASIC.)

FOR..DO is very similar to the FOR..NEXT loop in BASIC, and is illustrated in Listing 4. DO is usually followed by a compound statement, but single statements are acceptable. What is 'done' lies in the commands between DO and the first semicolon (which is at the end of line 12 in our example). From there, the program advances to the next command. Notice that if one enters '0' or '1' in this program as a response to line 8, the DO function will not be honored because it asks for a 'Y' greater than or equal to '2'. (The correct factorial will be printed in these cases, however.) It is possible to step down in a FOR..DO statement. The syntax in this instance, in line 11, would be....

'FOR X := 2 DOWNTO Y DO'.

A flexible and interesting branch command is the CASE..OF..ELSE command illustrated in Listing 5. Depending on the integer or string value of the expression following CASE, the line following OF which begins with this value will be executed to the first semicolon. An ELSE provision is available with this command also, as illustrated in line 13. Note that an END statement is required to mark the end of the CASE options (see line 12; it terminates the OF path, so to speak). Compare the punctuation in this listing and Listing 3.

```
SCR # 18
( *LISTING 4: FOR DO EXAMPLE )
PASCAL DECIMAL
PROGRAM FORLOOPEXAMPLE;
VAR
  X : INTEGER;
  Y : INTEGER;
  FACTORIAL : INTEGER;
BEGIN
  NEWLINE; WRITE ('ENTER NUMBER FOR FACTORIAL (<8) ');
  READ(#Y);
  FACTORIAL := 1;
  FOR X := 2 TO Y DO
    FACTORIAL := FACTORIAL * X;
  NEWLINE; WRITE ('FACTORIAL OF ', #Y, ' IS ' , #FACTORIAL);
END. ;S

```

```
SCR # 19
( *LISTING 5: CASE OF ELSE EXAMPLE ) PASCAL DECIMAL
PROGRAM CASEEXAMPLE;
VAR Y : INTEGER;
BEGIN
  NEWLINE; WRITE ( 'ENTER A NUMBER (0-2) ' );
  READ ( #Y ); NEWLINE;
  IF ((Y > -1) AND (Y < 3))
    THEN
      CASE Y OF
        0 : WRITE( 'THE NUMBER YOU ENTERED IS ZERO ' );
        1 : WRITE( 'THE NUMBER YOU ENTERED IS ONE ' );
        2 : WRITE( 'THE NUMBER YOU ENTERED IS TWO ' )
      END
    ELSE
      WRITE( 'SORRY, WRONG NUMBER! ' )
END. ;S
```

```
SCR # 20
( *LISTING 6: PROCEDURE EXAMPLE ) PASCAL DECIMAL
PROGRAM  PROCEDUREEXAMPLE;
VAR  X, Y, SQ : INTEGER;
PROC GETNO;
  BEGIN NEWLINE; WRITE('ENTER NO.(<182) TO FIND SQUARE, 0 TO STO
P '); READ(#Y); NEWLINE; END;
PROC SQUARE(U, V : INTEGER);
  BEGIN SQ := 0; U := 1;
   WHILE (U <= V) DO BEGIN SQ := SQ + V; U := U + 1 END;
  END;
PROC PRINTNO;
  BEGIN WRITE('THE SQUARE OF ',#Y,' IS ',#SQ) END;
BEGIN  \  MAIN  \
  REPEAT BEGIN GETNO; SQUARE(X, Y ); PRINTNO; END
  UNTIL SQ = 0;
END.  ( END MAIN )
```

SUBSCRIBER WRITES....

Subscriber Vance Pinter owns an IBM-XT he purchased to use in his daily office work and "hasn't regretted it for a minute." He included the color monitor to ease the transition from four years of CCII experience in his practice of law. The old machines are still used at home, for fun. Vance is especially interested in hardware articles and has added many options to his CCII, including lower case, the 4000H ram card by Tom Devlin, Frepost Computer's 16K RAM add-on, a bell, and second disk drive. He noticed an improvement from the disk drive modifications recommended by John Newman (Feb/Mar and Jun/Jul 1983).

"The Dynamic Ellipse Doodler by Tom Napier was fun. I keyed it in Sunday A.M. and it worked first try." Vance says he has thought of selling the old machines but just can't bring himself to do it. (I heard a whisper of regret from the Suits family not long ago that they no longer had the GOM on hand. ED)

CRT INTENSITY CONTROL

Instructions for mounting a front panel potentiometer to control CRT intensity have been prepared by subscriber Norman Johnson. A copy is available by writing to

The PROC (procedure) command is the single, most important concept in Pascal for creating a structured program. In a 'top down' structured programming approach, the functions of a program are separated into modules, each one programmed and tested separately, then joined together into one complete program. This removes complex debugging from program development, for if each parts works correctly, the whole will operate correctly.

To use a procedure, it must be defined before the program proper begins, and it may then be 'called' as many times as needed. Listing 6 demonstrates some sample procedures following the VAR and CONST commands. Here a procedure was written for each main part of the program. READNO reads in a number to be squared. SQUARE actually computes the square. PRINTNO prints the input number and its square to the screen. See how simple the main program becomes! The procedures are 'called' in the compound statement which follows the REPEAT command.

Note that the procedure SQUARE is the only one with a parameter list. In Pascal, this is a way of 'passing' a value or a number of values, back and forth between the main program and its procedures. The CALL to the procedure SQUARE, in the main program, also has a parameter list. The parameter list in the PROC and FUNC statements are optional, but if used, it must appear in both places. The names in the parameter list need not be the same, but their position in the list is important. This kind of parameter listing makes processes in the extended Pascal language 'portable' from one program to another, without renaming the variables they contain. Inserting variables in the right order is critical, however.

While full implementations of Pascal accommodate such a parameter list, in Tiny-Pascal, there is no provision for returning them. This is probably an oversight by its designer, Zimmer (or perhaps a deliberate part of the design!) So the result, SQ, is 'stuck' in the process. This really makes the

ANOTHER RETURN TO FCS

Chris Zerr submits his favorite way of returning to FCS or any other ESC [vector] from an assembly program. It is only applicable to v6.78 computers, however. If you first fill the A register with the vector code, such as [D] for FCS, [E] for Basic, [P] for 4000H, etc. If programs are ORiGined at 8200H they will be damaged by this method. The routine that completes the exit is located at 053AH in v6.78. A similar function has not been identified for v8.79 as yet. Here is a sample code:

```
START:  CALL    KEYIN   ;Get keyboard input
        CPI     'D'     ;Do some
        JZ      EXIT    ; error
        CPI     'E'     ; checking...
        JZ      EXIT
        ;
        ; [Error routines here]
        ;
EXIT:   PUSH    PSW     ;Save registers
        MVI     A,12    ;Clean up the screen
        CALL    LO
        POP     PSW     ;Restore registers
        JMP     053AH   ;Exit program
```

Colorcue. We advise that while the project is simple, it should not be undertaken by inexperienced hands. Damage to the CCII and/or injury through shock to the installer is a potential danger.

```
SCR # 21
  0 (  *LISTING 7: FUNCTION EXAMPLE 1  ) DECIMAL : CLS 12 EMIT ;
  1  0 VARIABLE FUNCTIONVALUE : VALTOSTACK FUNCTIONVALUE @ ; PASCAL
  2 PROGRAM FUNCTIONEXAMPLE; VAR X,A,B,C,Z:INTEGER;
  3 FUNC QUADRATIC(XVAL,AVAL,BVAL,CVAL:INTEGER);
  4   BEGIN
  5    FUNCTIONVALUE := AVAL*XVAL*XVAL+BVAL*XVAL+CVAL; VALTOSTACK;
  6   END;
  7 BEGIN  CLS; \ MAIN PROGRAM \
  8  REPEAT
  9   BEGIN NEWLINE; NEWLINE; NEWLINE;
 10    WRITE('ENTER X, A, B, & C TO EVALUATE (AX↑2+BX+C)*2+9 ');
 11    READ(#X,#A,#B,#C); NEWLINE; NEWLINE;
 12    Z := QUADRATIC(X,A,B,C) * 2 + 9;
 13    WRITE('THE Z VALUE IS ',#Z)
 14   END; UNTIL (X=0)
 15 END. ( MAIN )  DECIMAL  ;S
```

```
SCR # 22
  0 (  *LISTING 8: FUNCTION EXAMPLE 2  ) DECIMAL : CLS 12 EMIT ;
  1  0 VARIABLE FUNCTIONVALUE : VALTOSTACK FUNCTIONVALUE @ ; PASCAL
  2 PROGRAM FUNCTIONEXAMPLE; VAR X,A,B,C:INTEGER;
  3 FUNC QUADRATIC(XVAL,AVAL,BVAL,CVAL:INTEGER);
  4   BEGIN
  5    FUNCTIONVALUE := AVAL*XVAL*XVAL+BVAL*XVAL+CVAL; VALTOSTACK;
  6   END;
  7 BEGIN  CLS;   \ MAIN PROGRAM \
  8  REPEAT
  9   BEGIN  NEWLINE;  NEWLINE;
 10    NEWLINE; WRITE('ENTER X, A, B, & C TO EVALUATE AX↑2+BX+C ');
 11    READ(#X,#A,#B,#C); NEWLINE; NEWLINE;
 12    WRITE('THE QUADRATIC VALUE IS ',#QUADRATIC(X,A,B,C))
 13   END;
 14  UNTIL (X=0)
 15 END. ( MAIN )  DECIMAL  ;S
```

```
SCR # 23
  0 (  *LISTING 9: MEM[ ] EXAMPLE  ) : CLS 12 EMIT ;
  1 : HOME 8 EMIT 11 EMIT ; PASCAL DECIMAL
  2 PROGRAM POINTPLOT;
  3 CONST SCREENSTART = 28672;
  4 VAR X, Y, Z : INTEGER;
  5 FUNC PLOT(XVAL,YVAL:INTEGER);
  6   BEGIN
  7     IF ((XVAL>-1) AND (XVAL<64) AND (YVAL>-1) AND (YVAL<31))
  8     THEN MEM[SCREENSTART+X*2+Y*128] := 42
  9     ELSE BEGIN WRITE('RANGE ERROR '); Z := 1 END
 10   END;
 11 BEGIN  Z := 0; CLS;
 12   REPEAT BEGIN HOME; WRITE('ENTER X,Y ');
 13     READ(#X,#Y); PLOT(X,Y) END;
 14   UNTIL Z = 1
 15 END. ;S
```

parameter list for a PROC in Tiny-Pascal superfluous, and we suggest that you not try use one. It really isn't needed since the variables defined in the VAR statement, in the declaration part of the program, are 'global' type variables (that is; accessible to all parts of the program, as opposed to just a particular process.) So in Tiny-Pascal, the portability concept is lost, an unfortunate, but not serious, circumstance.

A subset of the PROC command, in Tiny-Pascal, is FUNC, the function command. For this command, we have devised a way to circumvent the direct inability to pass values from a procedure back to the main program. The 'function' is defined in Pascal to compute a single value, whereas a PROC may compute any number of values. The procedure SQUARE may be written as a function instead. Alas, in Tiny-Pascal, procedures and functions are defined in the same way, and we are stopped again from returning parameters. FORTH comes to the rescue here by allowing us to define a FORTH variable in which to store the value of a function, and then define a means of placing that value on the FORTH stack so that it may be used in an equation. The FORTH code to accomplish this appears in the second line of Listing 7.

To use a FUNC in this way (for otherwise it is the same as a PROC), simply code the second line of your screen in a fashion similar to Listing 7. (Those interested in understanding the commands may refer to the FORTH definitions in a suitable reference.) The function, like the procedure, is defined in the first lines of the program (line #1 of Listing 7.) Its purpose is to clear the CRT whenever it is called. Note that the program in Listing 7 requires four input values. Separate the entry of each value with a carriage return. (Commas must not be used to punctuate inputs to any of the programs in this article.)

In general, FORTH may be used to extend the power and versatility of Tiny-Pascal by using FORTH to define com-

```
SCR # 24
  0 ( *LISTING 10: IN[ ] EXAMPLE )
  1 PASCAL DECIMAL
  2 PROGRAM INTEST;
  3         VAR A3:INTEGER;
  4 BEGIN
  5         REPEAT READ(A3)
  6         UNTIL A3 IN[ 30+2,'Y' ];
  7         REPEAT READ(A3)
  8         UNTIL A3 IN[ 30+2,'E' ];
  9         REPEAT READ(A3)
 10         UNTIL A3 IN[ 30+2,'S' ]
 11 END.  ;S
 12
 13
 14
 15
```

```
SCR # 25
  0 (  *LISTING 11: ARRAY EXAMPLE  ) 0 VARIABLE N : COLOR N @ EMIT ;
  1 : BON 31 EMIT ; : BOFF 15 EMIT ; : CLS 12 EMIT ;
  2 PASCAL DECIMAL
  3 PROGRAM USEARRAY;
  4 VAR IA,IB:ARRAY[8] OF INTEGER;
  5     C,I,Z:INTEGER;
  6 BEGIN
  7  IA[ 1 ]:=16; IA[ 2 ]:=17; IA[ 3 ]:=18; IA[ 4 ]:=19;
  8  IA[ 5 ]:=20; IA[ 6 ]:=21; IA[ 7 ]:=22; IA[ 8 ]:=23;
  9  IB[ 1 ]:='B';IB[ 2 ]:=82; IB[ 3 ]:=71; IB[ 4 ]:= 'Y';
 10  IB[ 5 ]:=68; IB[ 6 ]:=80; IB[ 7 ]:=67; IB[ 8 ]:=87;
 11  CLS;
 12 WHILE C<>'F' DO
 13   BEGIN
 14     NEWLINE; NEWLINE; Z:=0; I:=1;
 15
```

mands which then become part of the FORTH/Tiny-Pascal dictionary. In addition, the core directory of FORTH, once mastered, may be referred to in creating special effects not obtainable otherwise through its own basic commands. This kind of language is said to be "extensible."

Listing 8 is a modification of Listing 7 showing an alternate way of referring to a FUNC so its output may be 'passed' back to the program.

Our next two commands are not present in extended Pascal. MEM is used in Tiny-Pascal to place a value in an absolute memory location. The program of Listing 9 uses MEM to 'plot' (as in Compucolor: PLOT 3) using screen memory. MEM uses the [ ] style of delimiter to contain the memory address. We set MEM[addr] to a legal value (using :=) between 0 and 255.

[Note: The MEM command did not function until screen #39 of the Tiny-Pascal compiler was altered. The '!' word, which follows the word COMPILE, must be changed to 'C!'. Note that CLS is also defined in FORTH, ahead of the Pascal program, to use later to clear the screen. If you haven't surmised already, 12 EMIT is the same as PLOT 12, in Basic. Note on the following line another FORTH word, 'HOME', is defined by using '8 EMIT 11 EMIT' (PLOT 8 PLOT 11), to return the cursor to the upper left corner and erase the line.]

The IN command allows one to check a single keyboard character for a match with a list of possibilities within [ ] delimiters. Only ASCII numbers or their single character string equivalents, enclosed in single-quotes, are permitted in the brackets. Listing 10 presents an example of this command in use. When the program is 'run', it screens the keyboard for the single characters 'Y', 'E', and 'S', in that order. Any other input is ignored.

Our last program example, the use of the integer array, is demonstrated in Listings 11 and 12. This is our only example employing two screens working together. This program makes liberal use of FORTH-defined commands, to execute the equivalent of BASIC's PLOT command. In the program, two parallel arrays are defined by assignment. One array contains single character strings or ASCII values, and the other contains the equivalent of BASIC's PLOT numbers. The user is asked to choose a color by selecting a single letter key. With a simple table 'look-up', controlled by a WHILE..DO sequence, the ASCII value of the letter is sought. When it is found, the value at the same index in the parallel array is used to set the color, prior to screen printout. Screen 25 has no final ';S' so the program continues on to the next screen.

Tiny-Pascal has provision for most commonly-used operators. These include MOD, NOT, OR, SHL, SHR (shift

CHIP REPAIR NETWORK FOR THE CCII

A network of private service facilities is being formed with the help of Intecolor Corporation, who have offered to assist us by supplying a parts inventory. So far, three locations have been designated. I have confirmation on only the first two listed here. You are invited to contact these service personnel regarding CCII repairs. All three come highly recommended, and all three have had extensive experience.

Steve Wooten: 155 Barington Street, Rochester, NY 14607. Telephone: (716) 442-4914. Call evenings.

Steve charges $20.00 per hour if he fixes your computer. There is no charge if he doesn't. Customer pays for parts and shipping both ways.

Gary Sipple: 27750 Golfview Street, Southfield, MI 48034. Write to Gary for details.

Bill Freiberger: Box 207, Mountain Lakes, NJ 07049. Telephone: (201) 263-2859 Write to Bill for details.

When having a CCII repaired the biggest hurdle is in the shipping. Severe damage has occurred in the past to computers improperly packaged. It is your responsibility to prepare the computer properly for shipment. Make certain that you consult with the repair agent you select before making shipment. The cathode ray tube, #13VAXP22, used in the CCII was proprietary and is no longer available!

```
SCR # 26
\ *LISTING 12: ARRAY EXAMPLE CONTINUED \
  WRITE('ENTER COLOR TYPE (B,R,G,Y,D,P,C,W)? F-END ');READ(C);
  WHILE Z<>1 DO
   BEGIN
    IF C=IB[I] THEN
     BEGIN N:=IA[ I ]; Z:=1
     END;
    I:=I+1; IF I>9 THEN Z:=1
   END;
   NEWLINE; NEWLINE; COLOR;
   IF I<10 THEN WRITE('THIS REPRESENTS THE COLOR CHOSEN ')
          ELSE
            BEGIN BON; WRITE('ENTRY ERROR '); BOFF
            END;
 END; C:='B'
END. DECIMAL ;S
```

bits left or right), 'less than' (and 'or equal to'), 'greater than' (and 'or equal to'), '=', '+', '-', '*', DIV, TRUE, and FALSE.

There are several commands in Tiny-Pascal we have not covered, such as TYPE and CALL. CALL is supposed to be a FORTH routine to 'call' non-alphabetic routines in Tiny-Pascal. Perhaps it will be the subject of a future article in DATA CHIP. The TYPE command does not function in the normal Pascal sense, so no immediate explanation of this command is available.

There is greater depth to Tiny-Pascal commands than has been conveyed in these articles. A more thorough explanation may be found in Jim Minor's Tiny-Pascal documentation in the PASCAL Syntax Diagrams. You may have copies of this material from the CHIP library.

This ends our Tiny-Pascal series in Colorcue. I apologize to those of you who have been 'stumped' by some features of Tiny-Pascal because of insufficient coverage here. Feel free to telephone me, evenings, at (716) 889-4994, if you would like to explore any particular feature in greater depth. My very best wishes to my friend, Joe Norris, and my heartfelt thanks to him for his contributions to the Compucolor Community by piloting Colorcue during its last year ☐.

Glen Gallaway 'found!'

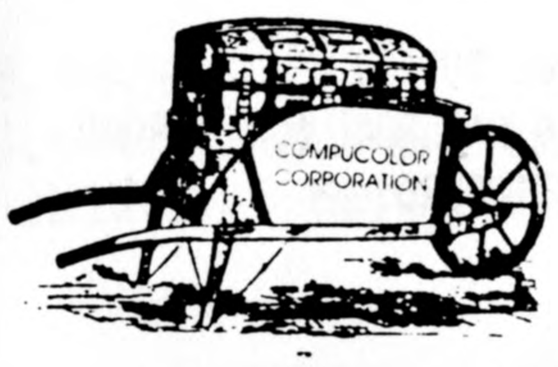
Intecolor 8000 users will be happy to know that Glen Gallaway, the 8000 coordinator, has been 'found' again, following some confusing address changes. His new address is 1637 Forestdale Avenue, Beavercreek, OH 45432. Glenn is relocated now and is looking forward to some permanence. Work continues to compile a useful set of memory addresses for conversion of CCII programs to the 8000 series computers. All 8000 users are invited to participate. Write to Glen at the above address if you are ready to help.

Tom Teaser!

Tom Napier wants to know if any readers can, without using the MVIinstruction, or making any preconditions, load the A-register with '06' using only 2 bytes!

TECH TIP, *DISK DRIVES*

One user had a problem with his disk drive that turned out to result from a drive belt near the breaking point. Doug Van Putte has found a source of drive belts for the CCII. Try Floppy Disk Services, 741 Alexander Road, Princeton, NJ 08540. Their phone number is (609) 799-4440. Ask for a standard 5-1/4" Siemans drive belt.



BACK ISSUES

Vol VI, numbers 1 and 2 are out of print. They will be available in Xerox form at $4.00 each if requested.

CHIP Library Reviews

We are interested in having your review of *CHIP* library holdings which you have found useful. Describe what the program does and how you use it. Include games, utilities, languages, etc.

COLORCUE CONTEST

Not one entry! The prize money will be returned to Colorcue's publication account. Shame! Shame!

SOFTWARE STILL AVAILABLE

You may still purchase CCII software from Intelligent Computer Systems in Huntsville, AL. The Muellers are permanently in the United States and are pursuing a diversity of activities involving computers and alternative energy sources. It is very considerate of them to maintain the CCII software library for us even though it must be more a nuisance than a business at this point. They are a good source for diskettes at a very low price.

CONNECTING EXTERNAL DRIVE

Gary Dinsmore has submitted plans for connecting an old internal drive as a second CCII drive. If you have such a drive available, Colorcue will be happy to send you a copy of his procedure for its installation.

NEXT ISSUE: CCII publications index; reviews of software by Bill Stanton and Wallace Rust; a captivating program adaptation by Tom Napier (a magnum opus 'Cuties'); an article on Recursion by Doug Van Putte; W. S. Whilly's final installment on debugging, a biographical sketch of Peter Hiner, and more. Get your contributions in promptly for our last issue!

CURRENT COLORCUE SUSCRIBERS

ACT DISTRIBUTOR
8522 CRANSTON DRIVE
WESTLAND, MI 48185

GEORGE R. ADAMS
4452 HIGHLAND CIRCLE
MARIETTA, GA 30066

JAY C. ALBRECHT
4686 FREEMAN ROAD
MIDDLEPORT, NY 14105

ED ALLEN
6040 SMITH LANE
REDDING, CA 96002

CHARLES L. ANDERSON
2728 HASTE STREET
BERKELEY, CA 94704

HAROLD L. ANDERSON
1106 WILSON STREET
RICHLAND, WA 99352

TOM ANDRIES
1625 N. IOWAS STREET
SOUTH BEND, IN 46628

B. J. ARBUNIC
5210 MARIT DRIVE
SANTA ROSA, CA 95405

GENE BAILEY
28 DOGWOOD GLEN
ROCHESTER, NY 14625

BEN BARLOW
161 BROOKSIDE DRIVE
ROCHESTER, NY 14618

MICHAEL P. BARRICK, VAL FRG HS
9999 INDEPENDENCE BOULEVARD
PARMA HEIGHTS, OH 44130

JOHN E. BEAM
4807 VALERIE
BELLAIRE, TX 77401

VINN BEIGH
1221 WEST GLENLAKE AVENUE
CHICAGO, ILL 60660

CREIGHTON BELL
3332 WAYSIDE
EL PASO, TX 79936

JOHN BELL
8300 4TH AVENUE
NORTH BERGEN, NJ 07047

WILLIAM R. BOCK
8317 LOUIS DRIVE
HUNTSVILLE, AL 35802

JACK H. BOGHOSIAN
1843 N. THORNE AVENUE
FRESNO, CA 93704

DAVID B. BOUCHER
33 ST JOSEPH AVENUE
FIRCHBURG, MA 01420

FATHER GEORGE BRUNISH
MATH DEPT/WESTMINSTER COLLEGE
NEW WILMINGTON, PA 16142

STEVE BRUNO
2626 CORONADO AVENUE
SAN DIEGO, CA 92154

R. S. BUCY
420 SOUTH JUANITA
REDONDO BEACH, CA 90277

MICHAEL R. BURCHAM
1707 GLEASON
IOWA CITY, IA 52240

ROBERT L. BURTON
114 AMANDA PLACE
OAK RIDGE, TN 37830

ARTURO L. CAZARES
2366 EAST FREMONT STREET
STOCKTON, CA 95205

JOSEPH J. CHARLES
130 SHERWOOD DRIVE
HILTON, NY 14468

DR. ORPHIA C. CHELLAND
501 NORTH PROVIDENCE ROAD
MEDIA, PA 19063

GENE COLLINS
ROUTE 7 BOX 354
CONWAY, SC 29526

PAT COLLEY
52 QUEENSWAY, CAVERSHAM PARK
READING, ENGLAND RG4 05J

COMMACK HIGH SCHOOL, MR WETJEN
VANDERBILT PARKWAY
COMMACK, NY 11725

VINCENT CORDOVA/NATIONAL MEDIC
PO BOX 433-A
WILLOW GROVE, PA 19090

BRIAN CRUSE
8 ULVA STREET, BALD HILLS
QUEENSLAND 4036, AUSTRALIA

MICHAEL DALEY
4643 CLOVER LANE
TOLEDO, OH 43623

WILLIAM L. DARKE
3310 SOUTH DEXTER STREET
DENVER, CO 80222

HUGH DARRAGH
32 McCAUL STREET/TARINGA
EAST BISBANE 4068 AUSTRALIA

S. DE SANTIS
700 AMSTERDAM ROAD
MT LAUREL, NJ 08057

MICHAEL DEVITO
2729 WIEDRICK ROAD
WALWORTH, NY 14568

JANE AND TOM DEVLIN
3809 AIRPORT ROAD
WATERFORD, MI 48095

THEODOR DIDRIKSSON
5916 AMAPOLA DRIVE
SAN JOSE, CA 95129

GARY A. DINSMORE
ROUTE 3 BOX 3216
WARREN, OR 97053

P. ALLEN DOW
260 NORTH MATHILDA AVENUE/A5
SUNNYVALE, CA 94086

PAUL DUDLEY
140 RAILROAD MILLS ROAD
PITTSFORD, NY 14534

RANDALL DUNSMOOR
615 SOUTH JACKSON STREET
GREEN BAY, WI 54301

RONALD EISENSMITH
22 KINGSWOOD
ORCHARD PARK, NY 14127

WILLIAM A. EMOND
1450 OAKLAND RD. SPACE #8
SAN JOSE, CA 95112

DUAINE ESBENSHADE
RIVER ROUTE, BOX 875
SILETZ, CA 97380

JOHN EWEL
DEPT OF BOTANY/UNIV OF FLORIDA
GAINESVILLE, FL 32611

MARK D. FAIRBROTHER
HC78 BOX 442 PENNVIEW APTS
BINGHAMTON, NY 13901

DR. MARJORIE FIRRING
8305 CHIANTI COURT
SAN JOSE, CA 95135

HOWARD FLANK/FLANK ASSOCIATES
2809 ATLANTA DRIVE
WHEATON, MD 20906

HENRY G. FLUCK
304 RANDLE COURT
CHERRY HILL, NJ 08034

J. FORD
PO BOX F
KIMMSWICK, MO 63053-0010

M. B. FRASER
1 LILY STREET/NORTH RYDE 2113
NEW SOUTH WALES, AUSTRALIA

BILL FREIBERGER
BOX 207
MOUNTAIN LAKES, NJ 07049

BRUCE A. GEIL/ G&B AUTO
7017 51ST AVENUE SOUTH
TAMPA, FL 33619

JAMES GICZKOWSKI
THE WURLITZER COMPANY BOX 591
CORINTH, MS 38834

IRVING GILES
172 WALNUT CREEK LANE
TOMS RIVER, NJ 08753

CHARLES H. GOULD
317 COCOA AVENUE
INDIALANTIC, FL 32903

D. B. GRANT
2 BROOKSIDE AVENUE/SOUTH PERTH
WEST AUSTRALIA 6151

EDWARD GREANEY
BOX 421 CHESTER AVENUE
NESHANIC, NJ 08853

DANIEL P. GREEN
905 BEECH
DINCAN, OK 73533

EVAN GREEN
11520 38TH NE
SEATTLE, WA 98125

WILLIAM L. GREENE
3601 NOBLE CREEK DRIVE, NW
ATLANTA, GA 30327

ART GRUSENDORF
BOX 605, MAGRATH, ALBERTA
CANADA TOK 1J0

FREDRIC HAERICH
1020 BROADWAY STREET
ALTAMONTE SPRINGS, FL 32714

JIM HALDEMAN
353 S WILLIE
WHEELING, IL 60090

ROBERT HALLEY
1714 MALDEN STREET
SAN DIEGO, CA 92109

CHARLES E. HAMMETT JR
611 BARKFIELD STREET
BRANDON, FL 33511

HAROLD R. HAMM
11739 LORETTO WOODS COURT
JACKSONVILLE, FL 32223

ROBERT G. HARDIN
1424 CHARLES AVENUE
KALAMAZOO, MI 49001

A. P. HARGREAVES
BRIDGE STREET/ ELTHAM
TARANAKI, NEW ZEALAND

BOB HARRIS
2954 SUNWOOD DRIVE
SAN JOSE, CA 95111

GLENN HAYHURST
9595 PECOS #14
DENVER, CO 80221

PETER HINER
11 PENNY CROFT/HARPENDEN
HERTS/ ENGLAND AL5 2PD

HARLEN HOWARD
832 SAN RAFAEL STREET
SUNNYVALE, CA 94086

FRED HUDSON
643 BROOKS ROAD
W. HENRIETTA, NY 14586

GRAHAM HUNT
17 BAROSSA CLOSE/ST CLAIR 2759
NEW SOUTH WALES, AUSTRALIA

INTEGRATED LOGISTICS SYSTEMS
PO BOX 518
ALTADÉNA, CA 91001

CHARLES IVY
10926 SHADOW WOOD DRIVE
HOUSTON, TX 77043

RONAN JAMES
11770 TIMBERLINE LANE
HALES CORNERS, WI 53130

NORMAN JOHNSON
STONE MARINA
JOHNS ISLAND, SC 29455

R. JONES
33383 LYNN AVENUE/ ABBOTSFORD
BRITISH COLUMBIA/CANADA V2S1E2

JAMES R. KENNEY
257 BERRY ROAD
BEAUMONT, TX 77706

JOHN KER
11839 WEST TRAIL
KAGEL CANYON, CA 91342

HARRY J. KEROP
76 RATTLING VALLEY ROAD
DEEP RIVER, CT 06417

KEN KERRISON
5 BELTANA ROAD/PIALLAGO A.C.T
AUSTRALIA 2609

ALDOLPH KLUKOVICK/KAY ENTERPR
2325 KINGSBRIDGE LANE
OXNARD, CA 93030

WILLIAM G. KNAPP
1761 CARMEL DRIVE
IDAHO FALLS, ID 83402

PAUL D. KOERBER
17090 SAN BRUNO/APT G23
FOUNTAIN VALLEY, CA 92708

FREDRICK G. KOMMRUSCH
5670 N. PARADISE LANE
MILWAUKEE, WI 53209

DR C. W. KREITZBERG
2311 N. FEATHERING ROAD
MEDIA, PA 19063

DENNIS L. LEPARD
120 S. ELLINGTON
DEPEW, NY 14043

A. LEWIS
PO BOX 228/ PARABURDOO 6754
AUSTRALIA

GOTE LILJEGREN
MARGARETAV 12 G/ 42 TABY
SWEDEN

FRANK M. LOCKE
2213 SOLORWAY
LAS CRUCES, NM 88001

ROBERT C. LOVICK
88 HILLHURST LANE
ROCHESTER, NY 14617

RIC LOWE
80 CAWSTON ROAD/ ATTADALE 6156
WESTERN AUSTRALIA

TERRY LUND
45 EVERGREEN STREET
SPENCERPORT, NY 14559

RONALD MACKENZIE
1 BRIAN STREET
COMMACK, NY 11725

GLEN MANN
6002 - 152ND AVENUE NE
REDMOND, WA 98052

JAMES MANUELLE
138 SHALE DRIVE
ROCHESTER, NY 14615

JOHN H. MASCARENHAS
PO BOX 78
LEDYARD, CT 06339

ROBERT H. MASKREY
1041 MILL ROAD
EAST AURORA, NY 14052

ALAN MATZGER
960 GUERRERO
SAN FRANCISCO, CA 94110

ANDY MAU
5 ELDRIDGE STREET
NEW YORK, NY 10002

PAUL F. MCCARRON
PO BOX 108
BELGRADE LAKES, ME 04918

FRED W. MCILROY, III
24426 24TH S STREET
KENT, WA 98031

MED/ASSIST DISTRICT AUDITORS
6815 WEST CAPITOL DRIVE
MILWAUKEE, WI 53216

ALAN MEGHRIG
2491 BUCKSKIN DRIVE E 13
LAGUNA HILLS, CA 92653

CAROLYN MEITLER
6110 MANSFIELD DRIVE
GREENDALE, WI 53129

BOB MENDELSON
27 SOMERSET PLACE
MURRAY HILL, NJ 07974

H. G. METZLER
235 BELCODA DRIVE
ROCHESTER, NY 14617

ALBERT J. MILLER
179 WALTER HAYS DRIVE
PALO ALTO, CA 94303

JACK E. MILLER
BOX 200
CARSON CITY, NV 89702

JOHN J. MINERD
BOX 191
YORKSHIRE, NY 14173

DR. JAMES MINOR
22 BRYN MAWR ROAD
ROCHESTER, NY 14624

GEORGE C. MOENCH, MD
1952 49TH STREET SOUTH
ST PETERSBURGH, FL 33707

THOMAS W. MONTEMARANO
1321 SWAN DRIVE
ANNAPOLIS, MD 21401

EARL H. MOORE
2112 BANCROFT
LAKE CHARLES, LA 70605

BRUCE R. MOREHEAD
3408 W. KIRBY
TAMPA, FL 33614

EIKE MUELLER
12117 COMANCHE TRAIL
HUNTSVILLE, AL 35803

TOM NAPIER
12 BIRCH STREET
MONSEY, NY 10952

JOHN E. NEWBY
4532 - 167TH AVENUE SE
ISSAQUAH, WA 98027

JOHN NEWMAN
8 HILLCREST DRIVE/ DARLINGTON
WESTERN AUSTRALIA 6070

DAVID C. NORMAN
3883 SAN MARCOS COURT
NEWBURY PARK, CA 91320

JOSEPH H. NORRIS
19 WEST SECOND STREET
MOORESTOWN, NJ 08057

LT COL DAVID NOWLIN
1103 SOUTH GRANDVIEW
PAPILLION, NEBRASKA 68046

H. T. ODUM
2106 NW 9TH AVENUE
GAINESVILLE, FL 32603

BRIAN E. O'HEARN
78 COLUMBUS AVENUE
SOMERVILLE, MA 02143

WILLIAM PARKER
2812 BERKLEY
FLINT, MI 48504

ASHOK S. PATWARDHAN
4260 CLAYTON ROAD APT #22
CONCORD, CA 94521

A.PAREIGIS
4411 SOUTH CROSS STREET
DOWNERS GROVE, IL 60515

STEVE PERRIGO
16925 INGLEWOOD ROAD NE B-306
BOTHELL, WA 98011

DULIN B. PERRY
506 HILLHURST DRIVE
BAYTOWN, TX 77521

LARRY W. PETERSON
6325 ELEANOR AVUNUE
OAKDALE, CA 95361

MICHAEL J. PETREYCIK
11 STAG LANE
TRUMBULL, CT 06611

MELVIN F. PEZOK
1381 IGNACIO BOULEVARD
NOVATO, CA 94947

ALEX V. PINTER, PC
PO BOX 230
COLUMBUS, GA 31902

RALPH J. PORTER
6157 SOUTH 700 W
MURRAY, UT 84107

WILLIAM J. POWER
8 PETER COOPER ROAD
NEW YORK, NY 10010

THOMAS PRICE, JR
129 HOMESTEAD AVENUE
DE BARY, FL 32713

J. RAMSEY
3115 S. ATLANTIC AVE #504
COCOA, FL 32931-2137

CARL E. REMLEY
149 BAYWOOD ROAD
BILOXI, MS 39532

HERBERT RICHARDSON
5805 ANTIONE ROAD
MOBILE, AL 36609

DAVID R. RICKETTS
108 BRYCE AVENUE
RED BANK, TN 37415

ALBERT RIDNER
CENTRO ATOMICO BARILOCHE
8400 BARILOCHE, ARGENTINA

JACK RIPPLE
65 APPLECROSS CIRCLE
CHALFONT, PA 18914

R. M. ROCKWELL
235 BELMONTE ROAD
WEST PALM BEACH, FL 33405

HERBERT ROSE/ WILCOMP OFF SVCS
465 WILSON AVENUE
DOWNSVIEW, ONTARIO M3H 1T9

HOWARD ROSEN
PO BOX 434
HUNTINGTON VALLEY, PA 19006

MICHAEL J. ROUSSE
15 SOUTH OWEN DRIVE
MADISON, WI 53705

WALLACE R. RUST
533 BRITTON ROAD
GREECE, NY 14616

BENJAMIN R. SAGE
16608 EAST STANFORD PLACE
AURORA, CO 80015

WILLIAM J. SEMBER
PHILIPS ECG, INC/JOHNSON ST
SENECA FALLS, NY 13148

ROY A. SHAFFER
530 SPRINGSIDE LANE
BUFFALO GROVE, IL 60090

WILLIAM SHANKS
1345 WEST ESCARPA STREET
MESA, AZ 85201

DONALD E. SHAUB
6294 HIGH STREET
EAST PETERSBURG, PA 17520

RONALD D. SHOOK
160 ANDOVER STREET
WILKES-BARRE, PA 18702

PHIL SIMON
6275 CARY AVENUE
CINCINNATI, OH 45224

GARY SIPPLE
27750 GOLFVIEW STREET
SOUTHFIELD, MI 48034

BOB V. SMITH
498 BROWN STREET
NAPA, CA 94558

BYRON E. SMITH
1189 LOCKE AVENUE
SIMI VALLEY, CA 93065

DAVID R. SMITH
5051 DIERKER ROAD / APT A4
COLUMBUS, OH 43220

MEL SMITH
NSSA/STDN
DANDAN, GU 96919

RALPH S. SMITH JR MD
5600 MAC CORKLE AVE SE SUITE 9
CHARLESTON, WV 25304

DR. JACK M. SPURLOCK
293 INDIAN HILLS TRAIL
MARIETTA, GA 30067

RICHARD F. SQUAILIA
818 MAIN AVENUE
SCHENECTADY, NY 12303

PETER STANDEN
12 KENDALL STREET/ CHARLESTOWN
AUSTRALIA 2290

BILL J. STANTON
8115 HELM COURT
COLORADO SPRINGS, CO 80918

DAN STIEFLER
S 3660 FULLER STREET
BLASDELL, NY 14219

JOSEPH STRATMAN
52029 US 33 NORTH
SOUTH BEND, IN 46637

DANA G/ STREBECK
8034 SHADY ARBOR LANE
HOUSTON, TX 77040

TED STUCKEY
BOX 420/ CAMBERWELL, 3124
VICTORIA, AUSTRALIA

DAVID SUITS
49 KARENLEE DRIVE
ROCHESTER, NY 14618

DICK SWARM
12709 GREENHALL DRIVE
WOODBRIDGE, VA 22192

JAMES P. SWEENEY
1566 WOMACK ROAD
DUNWOODY, GA 30338

ARTHUR TACK
1127 KAISER ROAD SW
OLYMPIA, WA 98502

BOB TALBOT
909 EXPLORER #2
RAPID CITY, SD 57701

RICK TAUBOLD
197 HOLLYBROOK ROAD
ROCHESTER, NY 14623

JOHN B. THIRTLE
105 COIFER LANE
ROCHESTER, NY 14622

MR. D. G. THOMAS
52 GROVE WAY/ WEMBLEY
MIDDLESEX, ENGLAND HA9 6JT

TOMMY THYSTRUP
VIRKELYST 20/ NR. SUNDBY 9400
DENMARK

J. P. TOOHEY
24 HOSKEN STREET/ NORTH BALWYN
MELBOURNE 3104/ AUSTRALIA

STEFFAN TOTH
PO BOX 1779, STN A
KELOWNA, BC V1Y 8P2

T. R. A. P. S.
PO BOX 8297 STATION "F"
EDMONTON, ALBERTA T6H 4W6

MARKE UNDERWOOD
1750 DYSON DRIVE NE
ATLANTA, GA 30307

DOUG VAN PUTTE
10 CROSS BOW DRIVE
ROCHESTER, NY 14624

CHRIS VERBEEK
14721 35TH AVENUE SE
BOTHELL, WA 98012

RICKI ANDREW VICK
702 WEST HOLLY AVENUE
STERLING, VA 22170

JAMES D. WARNER
11647 YUBA RIDGE DRIVE
NEVADA CITY, CA 95959

ROY WEISENBARGER
1201 CHESHIRE ROAD
MAITLAND, FL 32751

ROBIN WERNICK
9516 CARROLL CANYON DRIVE
SAN DIEGO, CA 92126

MAYNARD WILCOX
107 EAST AVENUE
FRANKFORT, NY 13340

MARC A. WILLIAMS
3 AMES STREET
CAMBRIDGE, MA 02139

WAYNE C. WILLIAMS, DIR
SCH OF MED/EAST CAROLINA UNIV.
GREENVILLE, NC 27834

WIS SURVEY RESEARCH LAB
610 LANGDON STREET/ 109 LOWELL
MADISON, WI 53703

THIMAS R. WOOLF
80 BOWEN ROAD
CHURCHVILLE, NY 14428

W. BRYANT WOOSLEY , JR
PO BOX 728
SHELBYVILLE, TN 37160

STEVE WOOTTEN
155 BARINGTON STREET
ROCHESTER, NY 14607

DAVID R. WRIGHT
1305 N EIRE / APT 23
LEXINGTON, NE 68850

WILLIAM B. WRIGLEY
4931 REBEL TRAIL, NW
ATLANTA, GA 30327

DAVID ZAWISLAK
5729 N CALIFORNIA AVENUE
CHICAGO, IL 60659

CHRIS ZERR
10932 - 156TH COURT NE
REDMOND, WA 98052

ANTHONY ZUVLIS
1117 SEQUOIA
FORT COLLINS, CO 80525

BASIC PRECISION

Subscriber Wallace Rust, of the Rochester User Group points out that Intecolor computers, including the CCII, can store numbers up to 16777216. While BASIC truncates and rounds them to six digits for display, it still stores them internally with at least 7-digit accuracy, and with 8-digit accuracy to the number stated above. You have nothing to lose, therefore, by entering constants with 8-digit accuracy. This will guarantee highest accuracy in computations before the rounding, prior to display, takes place. You can prove this rather easily with some simple experiments, perhaps using PI. [See some exploratory examples below. ED]

```
10   REM  Sample calculations to test numerical precision
20
30   X = 1.2345
40   PRINT 12 * X                              RUN
50
60   Y = 1.2345678                              14.914
70   PRINT 12 * Y
80                                              14.8148
90   PRINT 1/X
100                                             00.810045
110  PRINT 1/Y
120                                             00.81
130  PI(1) = 3.1416
140  PRINT PI(1) * 12^2                        452.39
150
160  PI(2) = 3.141592654                       452.389
170  PRINT PI(2) * 12^2
```

A 'no—echo' patch without assembly language!!

Here's one from Tom Devlin—a new version of the no-echo patch that uses no machine language! And it works!

Most users are familiar with the ESC USER jump location (if not, see Rick Taubold's detailed explanation in Colorcue). There is also a user-defined input flag vector which may be used to produce a deviation from the 'normal' flow of computer processes. Normally, the CCII performs its I/O with BASIC, FCS, the serial port, etc., according to default settings of the various flags. These settings are initialized by FCS at turn on time to prepare the computer for 'normal' operation. The programmer holds the power to alter this normal state of affairs. If the input flag at 33221-33223 (81C5H-81C7H), for example, is set to any of a group of special values, input is directed to a special jump vector (user defined), just like ESC USER. This jump vector can transfer operations to an unusual place, determined by the programmer.

As an example, the keyboard flag is at location 33247 (81DFH). This flag value is normally '0' and input keyboard characters are sent to the CRT. When it is '13' they go to FCS, which might send them back to the CRT or to a disk drive. When the flag is '12', the keyboard input is ignored. But if the flag holds any of the values 10, 11, 15-17, 19-22, 24, 26, or 28-31, then a jump to the memory location stored at address 33221—33223 occurs. If these addresses hold 8355H, for instance, then program execution continues at that location.

The routine below proceeds in just this way. The value '30' is placed in the jump vector address, 33221—33223. What we place in this address is the address of the routine that gets and stores the next keyboard press, but rather than allowing it to print, we trap these values for our own use.

Program lines 250-290 access the keyboard character, now in location 33278. From here on, Basic may do as it likes, we we have controlled the timing of this event to suit our needs. The following listing will demonstrate this technique for you. Type it in, and play with the variations that are possible. □

```
00010 REM    FIRST WE HAVE TO LOAD THE PROPER ADDRESS
00020 REM     INTO JUMP VECTOR "INPCRT" .THIS MUST BE
00030 REM     DONE BEFORE 'NO ECHO' IS REQUIRED
00040
00050 POKE 33221,195:REM POKE 'JMP' INTO FIRST BYTE
00060
00070 REM         FOR V6.78 THE ADDRESS IS 3FA0H
00080 IF PEEK (1)=108 THEN POKE 33222,163:POKE 33223,63
00090
00100 REM         FOR V8.79 OR V9.80 IT'S  0A01H
00110 IF PEEK (1)=186 THEN POKE 33222,1:POKE 33223,10
00120
00130 REM    FOR NO ECHO POKE 30 INTO "INPFLG" (33247),
00140 REM    TO RETURN TO ECHO POKE 12. (AN INPUT STATEMENT
00150 REM    OR THE END OF THE PROGRAM WILL ALSO DO IT)
00160
00170
00180 REM          THE FOLLOWING PROGRAM IS FOR DEMO ONLY
00190 REM          IT ECHOS THE ASCII EQUIVALENT OF THE
00200 REM          KEY YOU PRESS. PRESS "HOME" TO END
00210
00220 POKE 33247,30:REM TURN OFF ECHO
00230
00240
00250 POKE 33278,0
00260 K=PEEK (33278):IF K=0 THEN 260
00270 PRINT K;
00280 IF K=8 THEN END
00290 GOTO 250
```

CCII Color Adjustment

Wallace Rust

Compucolor users often have trouble distinguishing cyan from white, and yellow from green. The culprit is the green CRT phosphor, which emits not only green light, but also unwanted energy in the yellow, orange, and red regions of the spectrum. Thus, cyan images contain some unwanted red which makes them look white, and green images contain some unwanted red which makes them look yellow.

The trick to getting better colors is to adjust the green channel brightness so that its unwanted emissions are significantly weaker than the red emission of the red channel.

Page 6.06 of the Compucolor Maintenance Manual (ISC 216 978 999208) explains how to adjust the three screen grid controls R1 (red), R2 (green), and R3 (blue) located at the top of the circuit card at the base of the CRT. A similar procedure is outlined in the 3651 Manual, described here.

CAUTION: HIGH VOLTAGES ARE PRESENT IN THIS AREA OF THE COMPUTER. DO NOT ATTEMPT THESE ADJUSTMENTS UNLESS YOU KNOW EXACTLY WHAT YOU ARE DOING. NEVER USE BOTH HANDS TO MAKE ADJUSTMENTS. STAND ON AN ELECTRICALLY ISOLATED SURFACE WHEN WORKING ON THE COMPUTER. IF IN DOUBT, OBTAIN THE ASSISTANCE OF A QUALIFIED SERVICE PERSON.

(A convergence and color purity adjustment should be made first. Write to Colorcue for details of these adjustments.)

1. Erase the screen with a background color of white.
2. Turn potentiometers R1, R2, and R3 fully to their 'off' position. Turn the brightness control, mounted on the back of the unit, on the 50 pin bus side, below the FOCUS control, to maximum brightness.
3. Turn the red control, R1, until the red retrace area is just visable. Repeat the procedure for the green and blue controls (R2 and R3.)
4. Adjust the brightness control until there is no visable retrace line, and until brightness is at a comfortable level with a minimum of saturation.
5. Vary the adjustment of these controls to obtain the best white display at a comfortable level of brightness.

After following that procedure, do the following:

1. Put some test samples of each color on the screen at the same time.
2. Confirm that the red/blue ratio is such that you get a good magenta, which is neither too blue nor too red. Make minor pot adjustments to meet this criterium.
3. Turn R2 (green) to reduce the brightness of the green channel, until yellow becomes slightly orange (like KODAK yellow), and white becomes pink, relative to daylight white.

That's all there is to it!! ☐

UNCLASSIFIED ADS

FOR SALE: CCII, v6.78, lower case, 2 drives, full keyboard. Computer is in excellant condition, and comes with most issues of *COLORCUE* and *FORUM*, and many program disks. $1000.00. Call or write Bill Stanton, 8115 Helm Court, Colorado Springs, CO 80918; 303 599-4089.

FOR SALE: CCII, v6.78,32K, 1 drive. Maintenance Manual and following software: Personal Data Base, Basic Language 1-10, Formatter, Basic Editor, Screen Editor, Assembler, Fortran, miscellaneous games and disk utilities. All back issues of *COLORCUE*. Computer in good working order, $700.00, money order or certified check, prepaid. Bill Anthony, 655 Wells Way, Camano Island, WA 98292. 206 387-1576.

FOR SALE: CCII, v6.78, 32K, full keyboard, switchable lower case, handshake option, printer cable, sound, dust covers. Over 70 disks including Helm's Data Base, Screen Editor, Print; Com-Tronics TERMII; Income Tax Program, and many more. Manuals and periodicals included. $1500.00, prepaid. Phil Simon, 6275 Cary Avenue, Cincinnati, OH 45224. 513 681-8370.

BASIC VARIABLES IN FILE STATEMENTS

FILE "N" and FILE "R" statements in Basic use a combination of string and numeric parameters to specify the file name and attributes. It is not obvious that variables can be used to specify these quantities in the course of a Basic program. The examples below show the extremes to which variables may be used. Suppose you have a RND file of 128 bytes containing some data for each week in the year and want to read the file for a specific week. The following coding identifies the week number as a 'decimal' version number [1-52]:

```
420 INPUT "ENTER WEEK NUMBER > ";W$
430 FILE "R",2,"WEEK.RND;"+W$,1;1,128,1
```

String concatenation may be used to construct a proper file name as illustrated above. Note that the file name must be in string form. We could alternately call the RND files 'WK1.RND', 'WK23.RND', etc:

```
420 INPUT "ENTER WEEK NUMBER > ";W$
430 FILE "R",4,"WK"+W$+".RND",1;1,128,1
```

As long as we assign string variables where strings are required, and numeric variables where numbers are required, we can successfully operate on files with variables. Consider this example:

```
120 A$="WK" : C$=".RND" : F=1 : H=128
170 INPUT "ENTER WEEK NUMBER > ";W$
200 FILE "R",1,A$+W$+C$,1;F,H,F
```

A final example is taken from a working business program of mine (which has a well-documented file table in the manual!):

```
950 FILE "R",N,"CD"+D$+":"+Z$(2)+
    MID$(P$(2*CK),GH-5,T),F;4*P,KL-6,C
```

As always, a few 'controlled' experiments will help give you a confident grasp of the rules for using variables in this way. [JHN]

Assembly Language Programming

Part XVI

REPRINT TO CLARIFY FIG. 2

ANIMATION *(Concluded)*

CHRIS ZERR *10932-156th Court NE*
Redmond, WA 98052
Compuserve: 71445,1240

In the last article we plotted a rectangle on the screen and moved it through a blue field at various speeds using OSTR to print db strings.. Let's try a more appropriate figure this time, constructed from plot blocks, increase the action, and poke the animaton directly into screen memory.

We will construct an 'alien' with a missile launcher, moving feet, and piercing 'eyes' that blink menacingly at us, all through the use of plot blocks poked directly into screen memory. Constructing the figure of the alien is a matter of laying out the plot blocks and identifying the code required to illuminate them in the desired way.

CONSTRUCTING THE ALIEN'S PLOT BLOCKS.

Figure 1 shows a single plot block, consisting of eight controllable portions .. A1 through A4 and B1 through B4. These eight portions coincide with the eight bits of a byte. The portions of the plot block that will be illuminated will depend on whether or not the corresponding bit in the plot byte is set. To illuminate each of the four corners of a plot block in turn, we would set our plot byte equal to each of the following numbers:

```
Upper left corner : set A1 bit (bit 0)= PLOT value 1
Lower left corner : set A4 bit (bit 3)= PLOT value 4
Upper right corner: set B1 bit (bit 4)= PLOT value 16
Lower right corner: set B4 bit (bit 7)= PLOT value 128
```

So to construct a plot block, we shade in the bit sections, add all the individual bit values to the find the total plot value, and we have it! But we are not quite finished, because plotting the plot value alone will not quite do it. The CCI code must also be assigned, and for a plot block it must have a value between 128 and 255. The effect of the CCI code will be the same, by and large, as it would be for an ASCII character attribute, but the number will be increased by 128. For example, to set the color yellow, we issue PLOT 6,3 for ASCII, but for a yellow plot block it becomes 3 + 128 or 131. (Now we know how to peek at a screen memory location to tell if there is an ASCII character there or a plot block.)[1]

You may easily experiment with plot blocks and their CCI codes in BASIC to observe the diffference various CCI codes make. Whatever character we design must be derived from a combination of plot blocks within the possibilities available to us. A crude sketch on graph paper is the easiest way, possibly, and permits an instant translation into the correct plot codes.

Fig 1.

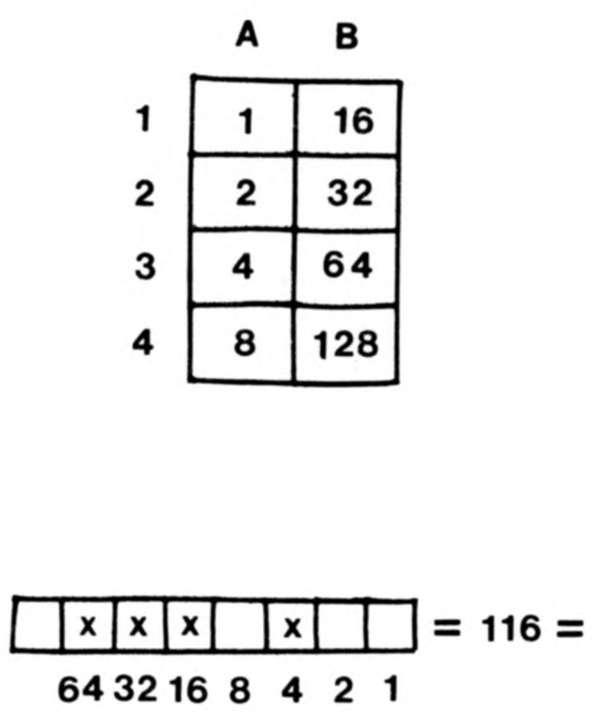
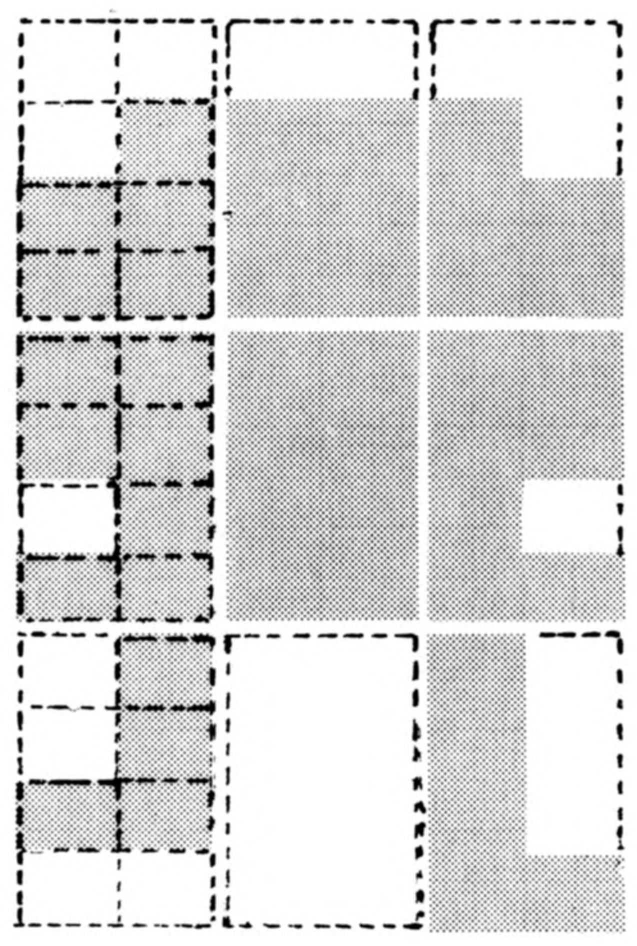


Fig 2.



Our alien has two different forms in this program. In the first form his left foot is raised; in the other his right foot is raised. Figures 2 and 3 illustrate the composition of these two figures. The outline of the plot blocks has been exaggerated in Fig 2 so you can see how the plot codes for this figure were derived. The upper left plot block has bits 2, 3, 5, 6, and 7 set, giving a value of 4 + 8 + 32 + 64 + 128 or 236 for this plot block. You should try to derive the plot codes for the remaining blocks as an exercise. The lower left block is shown with the proper bits set for you in Figure 2.

The plot blocks from Figure 2 contain the values 236, 238, 206, 251, 191, 116, 0, 143. The zero (0) count is necessary to act as a filler which says a null block is to be plotted in that location. In actual practice, each of these plot blocks must also have a CCI code following it, so a completed command byte string for Figure 2 would be 236, 131, 238, 131, 206, 131, etc.

We may mix plot blocks with ASCII characters in the plot command string, and our alien requires this to plot his 'eyes', which are quotes in ASCII. You will find them in the plot string from this Basic example, which plots the alien in one of his two forms on the screen:

```
100  X=30780 : REM INITIAL SCREEN MEMORY LOCATION
110  FOR A=1 TO 3 : REM PAINT 3 VERTICAL PAIRS OF BLOCKS
120  FOR B=1 TO 6 : REM THREE PAIRS OF BYTES FOR X AXIS
130  READ Z : REM GET POKING DATA
140  POKE X,Z : REM POKE ALT PLOT CODES & CCI
150  X=X+1
160  NEXT B
170  X=X+122 : REM PLOTS DOWN ONE BLOCK ON Y AXIS
180  NEXT A : REM PLOT NEXT VERTICAL PAIR
190  END
200  DATA 236,131,238,131,206,131,251,131,34,25,191,131
210  REM '34,25' IS ASCII ["] IN RED
220  DATA 116,131,0,0,143,131 : REM '0,0' IS FILLER PAIR
```

WELCOME ALL TO

```
 _____   _    __    __    ___   ___
   |    / \   | \   | \    |   /   \
   |    |___| |  |  |  |   |   \__
   |    |   | |__/  |  |   |       \
   |    |   | |  \  |__/  _|_  \___/
```

THE INTECOLOR BULLETIN BOARD

(206) 483 - 3460

This BBS is called TARDIS, run by the 'Doctor and Dee' at 206-483-3460. It may be accessed at 300 baud by TERMII (Com-Tronics) or any 'standard' terminal emulation software that might be used to talk to any other network. TARDIS is hardly confined to Intecolor products. In fact, we are new entrants. TARDIS services CP/M, OS-9, IBM-PC, Flex, Commodore 64 and others as well.

It is an interesting little board, and I suspect that there can be some unique rewards from using it. The 'Doctor' knows his stuff, and has a great familiarity with micros and micro software. He will try to get any public domain software you might need.

GETTING STARTED. This can be a little tricky! Dial the BBS. When the carrier is detected, type <RET>. A blank screen will stare at you. The message 'Nulls, if needed, (0-9)?' will appear. If nothing is happening, type any number from '0' to '9' and the logon message will appear. You will be asked for your first and last names, and a code word to identify yourself, each of these prompted by the BBS.

As a first-time user, you have few priveleges. You will be given a series of options, 'B,E,R,S,K,G,W,J,U,T,X,P,C,N (or ?):' to select from. Type '?' for HELP and to see what's going on. As for the others, here is their meaning:

<B> Bulletin File	<C> Chat with the Doctor
<E> Enter Message *	<G> GO 'Leave the system'
<R> Read Message	<S> Scan message base
<K> Kill Message	<U> Users file (Members)
<T> Toggle BELL	<L> List callers *
<W> Welcome Message	<P> Password Change
<J> CP/M system %	<X> X'pert User

* Special function requirements. Verified users only.
% CP/M is open to those people who leave information on themselves.

You will do best to choose 'G' to leave the system. As crazy as this sounds, your first visit is a little 'suspect' to the system, for reasons any BBS user already knows. To make yourself a respectable member of the community, you must first tell a little about yourself. After choosing 'G', you will see the following two lines -

Enter confidential comments for the Doctor (Y or N) _
(type 'Y') ... and
Enter text; type two RETURNs to end.

At this point proceed with care! WAIT for the '>' prompt to appear before you begin typing your introductory message to the 'Doctor.' Each line will hold only about 60 characters and must be terminated with a RETURN. Now you must wait a second or so for the '>' prompt to reappear to receive your next line of text. You must not try to enter a blank line, for that constitutes 'two RETURNs' and you will be terminated.

After pressing two RETURNs to 'end', you will be asked to turn off your modem. This is an inelegant but effective way of stopping your long distance charges.

If the 'Doctor' is able to read your message, he will, theoretically, install you as a 'member in good standing', and your second visit will be more fruitful. As with all early BBS accesses, you will be wise to activate a log file for the session (turn on your buffer in TERMII) and save it to disk before quitting your terminal software. This way you can review all that happened in your leisure, and make subsequent visits more fluent and less expensive.

COLORCUE readers have proven themselves to largely a 'lazy', quiet collection of people. Our attempt at making 'communication' a theme for Volume VI has not been fruitful. But for those of you who like to have fun and share, this is a new opportunity. I urge you to be active with TARDIS, and leave a few messages from time to time. You may advertise your own software, tell about special programs or hints you have available, ...whatever. We need to keep all our communication channels busy, and TARDIS can be a fast-access and helpful supplement to CHIP as a news medium for CCII users.

As always, call or write to COLORCUE if we can be of service.